UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

APLICACIÓN MÓVIL PARA LA CAPTURA GUIADA DE IMÁGENES PRECISAS DE OBJETOS PARA SU ANÁLISIS DETALLADO

Jorge Gómez Galván
Tutor: Jesús Bescos Cano

SEPTIEMBRE 2019

APLICACIÓN MÓVIL PARA LA CAPTURA GUIADA DE IMÁGENES PRECISAS DE OBJETOS PARA SU ANÁLISIS DETALLADO

AUTOR: Jorge Gómez Galván
TUTOR: Jesús Bescos Cano

Video Processing and Understanding Lab
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Septiembre de 2019

Resumen

Este Trabajo de Fin de Grado ha consistido en el desarrollo de una aplicación móvil para dispositivos Android, basada en la detección de objetos previamente modelados mediante su captura, tanto de forma automática como manual, para su posterior almacenamiento en la memoria del teléfono móvil y análisis.

Primeramente, se ha realizado una explicación de las tres tecnologías empleadas y el punto desde donde partía este proyecto. En el siguiente paso se han definido los requisitos que se han de cumplir, la estructura de la aplicación y el software a utilizar para el desarrollo de esta.

Posteriormente, se ha iniciado el desarrollo de la aplicación a partir de la creación de una interfaz gráfica lo más práctica e intuitiva posible para los usuarios. Consecutivamente, se ha expuesto el funcionamiento de la captura de imágenes mediante un algoritmo de detección implementado con técnicas de tratamiento de imágenes, así como el almacenamiento de las fotografías realizadas en cada uno de los dos modos de captura. A continuación, para poder visualizar las imágenes guardadas, se ha ofrecido la posibilidad de acceder desde la propia aplicación a la galería del dispositivo. Para concluir el desarrollo, se han modelado e incluido nuevos objetos en la aplicación.

Con el propósito de verificar la eficiencia de la aplicación, se han realizado numerosas sesiones de captura para obtener y examinar detalladamente el comportamiento del algoritmo de detección en diversos entornos con condiciones favorables y adversas en el proceso de reconocimiento de objetos. Finalmente, se han analizado las conclusiones a las que se han llegado y se han planteado distintas propuestas para continuar el trabajo en un futuro.

Palabras clave

Aplicación, Android, detección, objetos, silueta, máscara.

Abstract

This Bachelor Thesis has consisted in developing a mobile application for Android devices, based on objects' detection. These objects were previously modeled by capturing them, both automatically and manually, and in this way, they can be stored in the mobile phone's memory and analyzed.

Firstly, the three technologies used along this project and the starting point of this thesis were explained. The following steps have been determining the requirements, the application's structure and the software employed for developing this application.

Subsequently, the development of the application has been initiated by creating a graphical interface, remarkably practical and intuitive for the users. Consequently, the image capture operation has been exposed through the implementation of a detection algorithm using imaging techniques and, also, the storage of the photographs taken in each of the two capture modes. To follow with, to be able to view the saved images, the possibility to access the device´s gallery has been offered directly from the application. As a conclusion of the development, new objects have been modeled and included in the application.

In order to verify the efficiency of the application, numerous capture sessions have been carried out for obtaining and examining in detail the behavior of the detection algorithm in various environments. Those environments presented favorable and adverse conditions for the objects' recognition process. To end up with, conclusions reached have been analyzed and several proposals have been suggested as future directions for this project.

Keywords

Application, Android, detection, objects, silhouette, mask.

Agradecimientos

Quiero agradecer, en primer lugar, a Jesús Bescos por darme la oportunidad de realizar este Trabajo de Fin de Grado y prestarme su ayuda en todo momento.

También quiero dar las gracias a mis familiares, a mi pareja, a amigos y a mis compañeros de universidad. Vuestro apoyo y vuestra ayuda han sido esenciales durante el grado.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Motivación

En los últimos años, los teléfonos móviles han sufrido un enorme avance que ha provocado un importante impacto a nivel tecnológico en la sociedad actual. Su alto crecimiento no solo ha revolucionado la manera de entender la comunicación, sino que también ha influido en la simplificación de cualquier actividad.

El continuo auge de estos dispositivos ha provocado un aumento exponencial en el uso de aplicaciones, lo que ha implicado que muchas de ellas se hayan convertido en una herramienta esencial para multitud de campos, entre los cuales se encuentra el empleo de técnicas de visión por ordenador. En esta disciplina, que incluye métodos para obtener, procesar y examinar imágenes [1], es donde se encuadra el reconocimiento de objetos. Uno de los retos de este desafío consiste en el análisis preciso de objetos para extraer medidas cuantitativas sobre sus características.

La motivación fundamental del presente proyecto se centra en el desarrollo de una aplicación móvil que permita detectar determinados objetos y realizar fotografías de estos con la cámara integrada para su posterior análisis.

1.2 Objetivos

En este Trabajo de Fin de Grado se pretende llevar a cabo el diseño y el desarrollo de una aplicación móvil que guíe al usuario en la captura de ciertos objetos, tanto de forma automática como manual, utilizando técnicas de tratamiento de imágenes. Además, ofrecer desde la propia aplicación el visualizado de las imágenes capturadas con la finalidad de realizar diferentes acciones con ellas.

En primer lugar, para que la utilización de la aplicación sea lo más satisfactoria posible se ofrece una interfaz gráfica sencilla e intuitiva para los usuarios. Por consiguiente, se necesita familiarizarse con el entorno de desarrollo oficial de aplicaciones para dispositivos Android.

Asimismo, es esencial para el proyecto iniciarse en el uso de la biblioteca de código abierto OpenCV para manipular las imágenes captadas por la cámara a través de ciertos procesos y así implementar un algoritmo de detección de objetos. Para ello, se requiere establecer varios modelos de objetos y, a partir de ahí, realizar un análisis básico de la imagen enfocada mediante un proceso interactivo hasta obtener la imagen deseada.

Finalmente, se exige la toma de fotografías de calidad a pesar del entorno donde se encuentre el usuario en el proceso de captura. Con el fin de comprobar la eficiencia del algoritmo de detección, es necesario la realización de numerosas sesiones de captura de imágenes en varios entornos con condiciones dispares.

1.3 Organización de la memoria

La memoria de este trabajo consta de los siguientes capítulos:

- **Capítulo 1:** Introducción.
- **Capítulo 2:** Estado del arte.

- **Capítulo 3:** Diseño.
- **Capítulo 4:** Desarrollo.
- **Capítulo 5:** Pruebas y resultados.
- **Capítulo 6:** Conclusiones y trabajo futuro.

2 Estado del arte

Antes de comenzar con la explicación del diseño y del desarrollo de la aplicación móvil conviene realizar una breve descripción de las tecnologías de partida empleadas, así como el punto de partida de este proyecto.

2.1 Tecnologías de partida

2.1.1 Android Studio

Android Studio [2] es un entorno de desarrollo integrado que proporciona una gran cantidad de herramientas para el desarrollo de aplicaciones para la plataforma Android. Este entorno está basado en el programa informático IntelliJ IDEA y, además del editor de código y las mencionadas herramientas de desarrollo, cuenta con el sistema de compilación Gradle, un emulador para ejecutar y probar la aplicación, y un entorno unido para desarrollar en todas las versiones de Android. Además, es compatible con C++ y NDK.

Todo proyecto tiene uno o más módulos que proporcionan archivos de código fuente y archivos de recursos. Estos módulos incluyen módulos de aplicaciones de Android, módulos de biblioteca y módulos de Google App Engine.

En cada módulo de la aplicación se encuentran las siguientes carpetas: *manifests*, que contiene el archivo de manifiesto (*AndroidManifest.xml*), que facilita información fundamental sobre la aplicación; *java*, que contiene archivos de código fuente, en los que se incluye el código de prueba Junit; y *res*, la cual ofrece todos los elementos utilizados por la aplicación, como imágenes o diseños XML que representan gráficamente las actividades.

Android Studio incluye un sistema de compilado potente y de fácil uso basado en Gradle, que aumenta la productividad desde el inicio. La configuración del compilado y todas las dependencias del proyecto se almacenan en los archivos de Gradle (*build.gradle*), generados automáticamente por Android Studio. Cada aplicación contiene un archivo de nivel superior para todo el proyecto y uno de nivel de módulo independientes por cada módulo existente.

2.1.2 OpenCV para Android

OpenCV (Open Source Computer Vision Library) [3], [4] se trata de una biblioteca de software de código abierto de visión artificial, inteligencia artificial y aprendizaje automático. El objetivo de su creación fue proveer un conjunto de servicios comunes para las aplicaciones de visión artificial y activar la utilización de la percepción de la máquina en el comercio.

La librería cuenta con más de 2.500 algoritmos optimizados que son empleados para la detección y reconocimiento de rostros, la identificación de objetos, la clasificación de acciones humanas en vídeos, el rastreo de objetos en movimiento, la extracción de modelos 3D de objetos, la búsqueda de imágenes similares en una base de datos, el seguimiento de los movimientos de los ojos, etc.

El paquete que posibilita el desarrollo de aplicaciones de Android con las librerías de OpenCV es OpenCV4Android SDK. También es necesario la instalación de la API de

OpenCV Manager en el dispositivo Android para acceder a la biblioteca OpenCV. Dicha interfaz de programación de aplicaciones ofrece una serie de ventajas a los desarrolladores: tamaño compacto del APK, optimizaciones específicas de hardware, actualizaciones automáticas y corrección de errores.

2.1.3 MATLAB

MATLAB (MATrix LABoratory) [5] es un entorno interactivo para desarrollar algoritmos, analizar y visualizar datos, y realizar cálculos numéricos. Es posible importar datos a MATLAB desde archivos, otras aplicaciones o dispositivos externos. Una vez que los datos se encuentran dentro de la plataforma, se pueden explorar y analizar mediante funciones matemáticas y de ingeniería integradas, así como por medio de gráficas y visualizaciones.

La base de MATLAB es su propio lenguaje de programación, que soporta operaciones con matrices y vectores que resultan fundamentales para resolver problemas de ingeniería y científicos. Los comandos se pueden ejecutar de uno en uno, obteniendo así resultados inmediatos, o también a través de uno o varios archivos de script, lo que permite explorar diversos enfoques y realizar iteraciones hasta alcanzar una solución óptima.

Las herramientas de desarrollo permiten implementar los algoritmos de forma eficiente y optimizar su rendimiento. MATLAB proporciona las características de un lenguaje de programación tradicional, además de herramientas de presentación para diseñar interfaces gráficas de usuario personalizadas. Los toolboxes complementarios amplían el entorno de MATLAB para resolver problemas en diversas aplicaciones, como el tratamiento de señales y comunicación, el tratamiento de imagen y vídeo o el diseño de sistemas de control, entre otros.

2.2 Punto de partida

Este proyecto parte de un Trabajo de Fin de Grado previo cuyo resultado final consiste en una aplicación básica de Android que guía la captura, con la cámara del teléfono móvil, de un objeto previamente modelado. La aplicación facilita que se reconozca el objeto y se tome de manera automática una foto de la imagen captada por la cámara en el momento en el que la silueta del objeto elegido se superponga sobre la del objeto captado por la cámara del dispositivo.

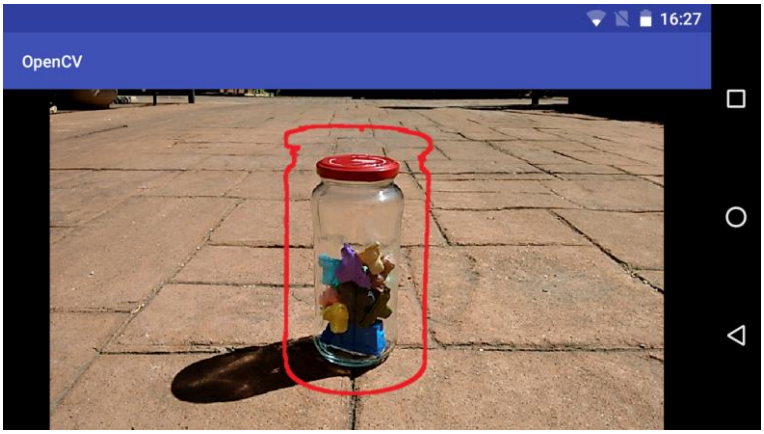

Figura 2-1: Aplicación inicial

Para ello, se comenzó restringiendo que el objeto a detectar fuese un tarro de cristal. Posteriormente, se creó su silueta en color rojo mediante el uso del algoritmo de Canny con el propósito de que el usuario sea guiado en la captura del objeto concreto. Finalmente, se implementó un algoritmo básico para la detección del objeto deseado basado en el conteo de píxeles de la silueta creada con la obtenida a través de la cámara.

3 Diseño

En el siguiente capítulo se presentan las funcionalidades y la estructura de la aplicación desarrollada, así como los paquetes de software empleados durante el proyecto.

3.1 Especificación de requisitos

Las pautas a seguir para el desarrollo de la presente aplicación móvil son las que se enumeran a continuación:

- Una pantalla de inicio que contenga tres botones para acceder al capturado automático, al capturado manual y a la galería.
- Un modo que tome fotos de manera automática una vez detectado el objeto deseado. Para ello, la cantidad de píxeles blancos calculados mediante el algoritmo de detección debe superar el valor de un umbral decidido.
- Un modo que permita tomar fotos de manera manual, modificando el color de la silueta una vez detectado el objeto deseado. La condición que ha de cumplirse para dicho cambio de color será similar a la del capturado automático.
- Una galería que ofrezca el visualizado y el almacenamiento en la memoria del dispositivo móvil de las imágenes de los objetos capturados y las máscaras utilizadas en cada captura. Además, proporcionará la posibilidad de enviar por correo o por otras aplicaciones instaladas las imágenes guardadas por la aplicación.
- Un menú que posibilite al usuario cambiar, tanto en el capturado automático como en el manual, la silueta del objeto que se quiere reconocer. Con ese fin, será necesario la creación de nuevas siluetas.
- Un icono y un nombre propio para la aplicación, así como establecer y ajustar los temas en cada una de las actividades.

3.2 Estructura de la aplicación

Uno de los objetivos más relevantes durante este proyecto ha sido el proporcionar una interfaz simple e intuitiva para los futuros usuarios. Es por eso por lo que antes de iniciar el desarrollo en Android Studio se estructuraron las diferentes vistas que forman la aplicación.

Como se muestra en la siguiente figura, la aplicación consta de cuatro partes diferenciadas.

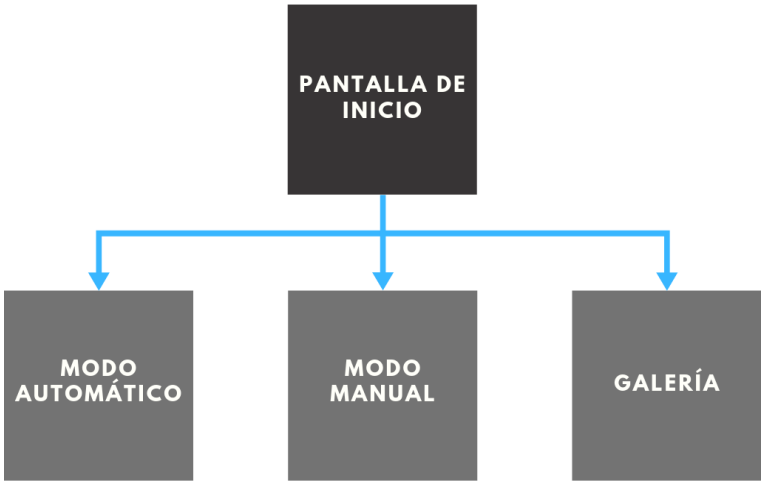

Figura 3-1: Diagrama de la estructura de la aplicación

3.2.1 Pantalla de inicio

La pantalla de inicio es la primera actividad que se ejecuta al arrancar la aplicación. En ella, se encuentran los controles en forma de botón que posibilitan al usuario desplazarse por el resto de las actividades.

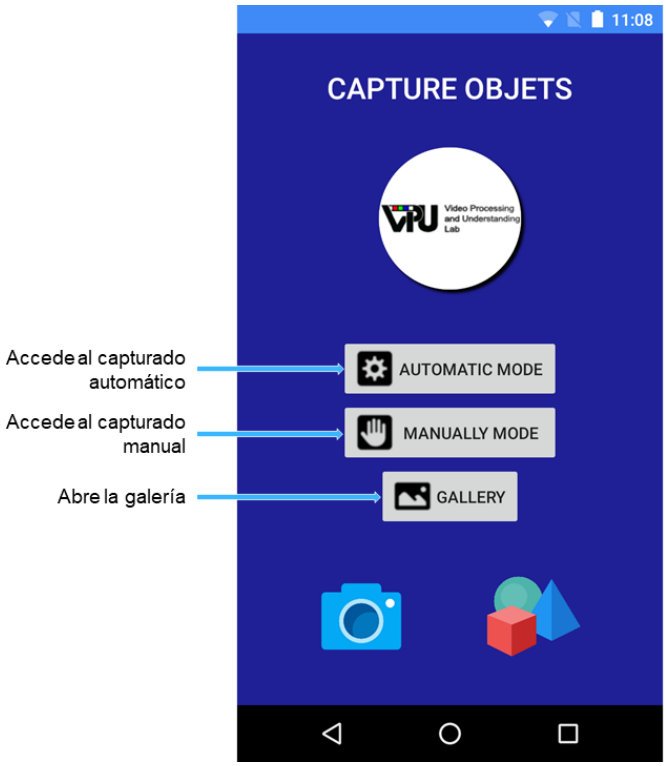

Figura 3-2: Pantalla de inicio

3.2.2 Modos automático y manual

Tanto el modo automático como el manual poseen una vista prácticamente idéntica y comparten las funcionalidades indicadas en la figura inferior. Sin embargo, el resultado de sobreponer la silueta de color rojo a la del objeto es diferente en cada modo.

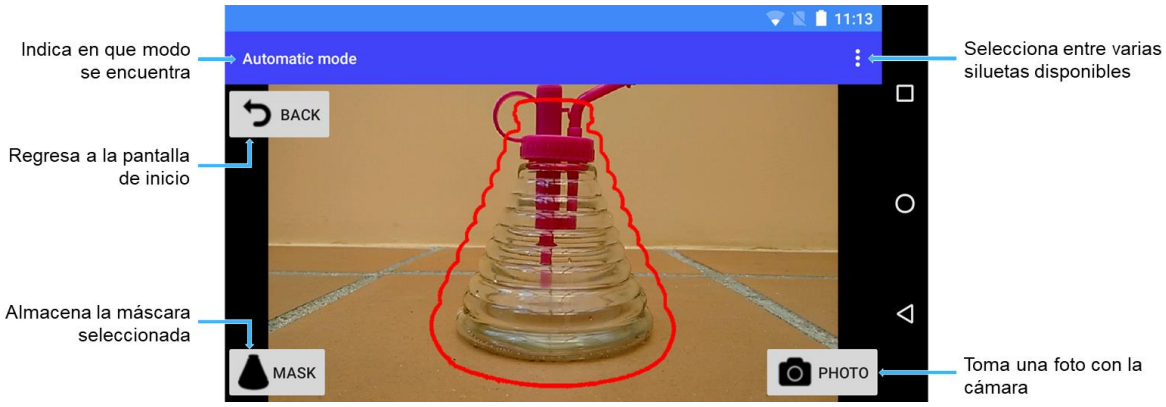

Figura 3-3: Modo automático

3.2.3 Galería

A través de la galería se pueden visualizar las fotografías tomadas por la cámara del teléfono móvil y las distintas máscaras guardadas. También incluye diversas funciones que permiten compartir, editar y eliminar las imágenes.

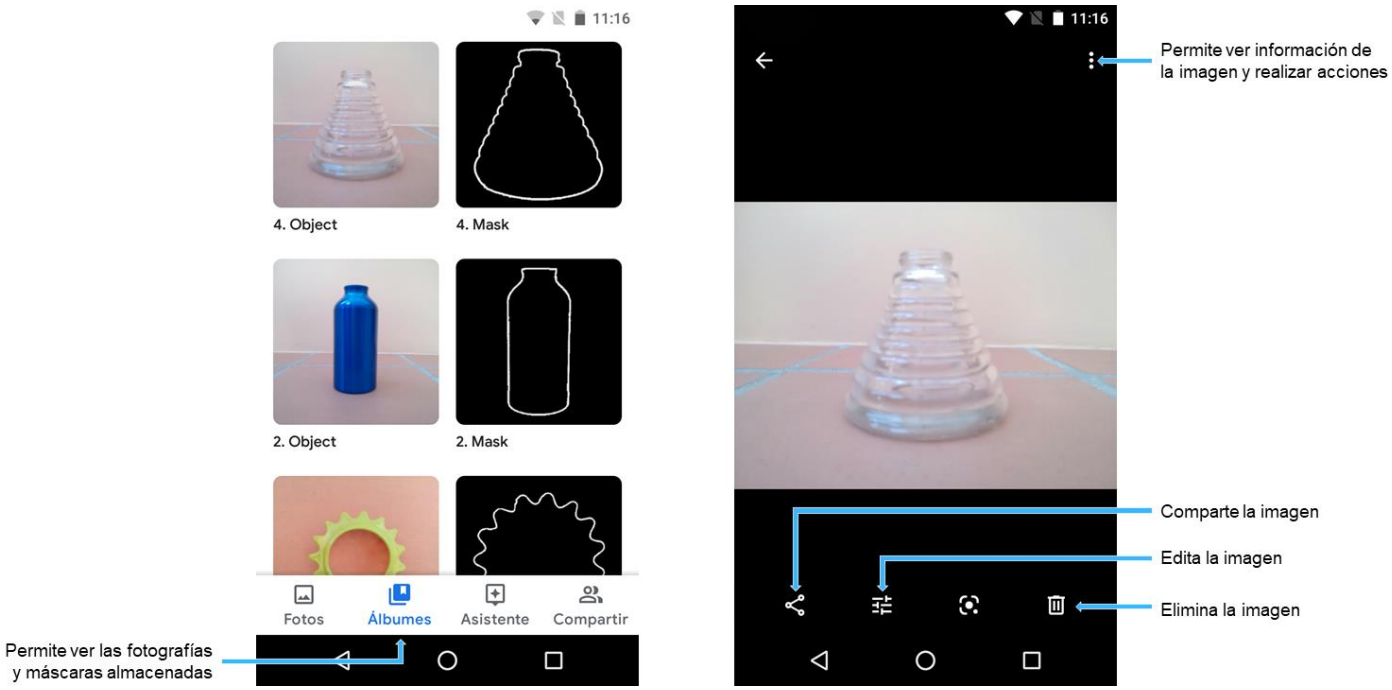

Figura 3-4: Galería

3.3 Paquetes de software empleados

La primera cuestión a considerar a la hora de iniciarse en el desarrollo de una aplicación móvil es la elección del sistema operativo. En la sociedad actual impera principalmente el uso de dos: Android e iOS.

Uno de los principales argumentos por los que se ha escogido el primero ha sido por su lenguaje de programación. Android utiliza Java, uno de los lenguajes más populares en la actualidad, mientras que Apple utiliza Objective-C/Swift, su propio lenguaje.

Otra diferencia notable se encuentra en la fase de pruebas, clave para comprobar el perfecto funcionamiento de la aplicación. El simulador de iOS es mucho más rápido que el emulador de Android. Sin embargo, Android proporciona una simulación más realista y una mayor facilidad para realizar pruebas en dispositivos reales [6], aunque la variedad de dispositivos que utiliza Android es tan amplia que se necesita adaptar la aplicación a cada modelo de pantalla para que la experiencia de usuario sea lo más satisfactoria posible.

Sin embargo, a pesar de los dos puntos anteriores, conviene conocer el número de usuarios potenciales que vamos a encontrar en cada uno de los sistemas operativos. En este sentido, Android también domina claramente a iOS. Según los datos extraídos de StatCounter [7], Android lleva dominando el mercado de sistemas operativos en dispositivos móviles más de cinco años, con unas tasas de crecimiento muy elevadas hasta 2016, año en el que se ha estabilizado.

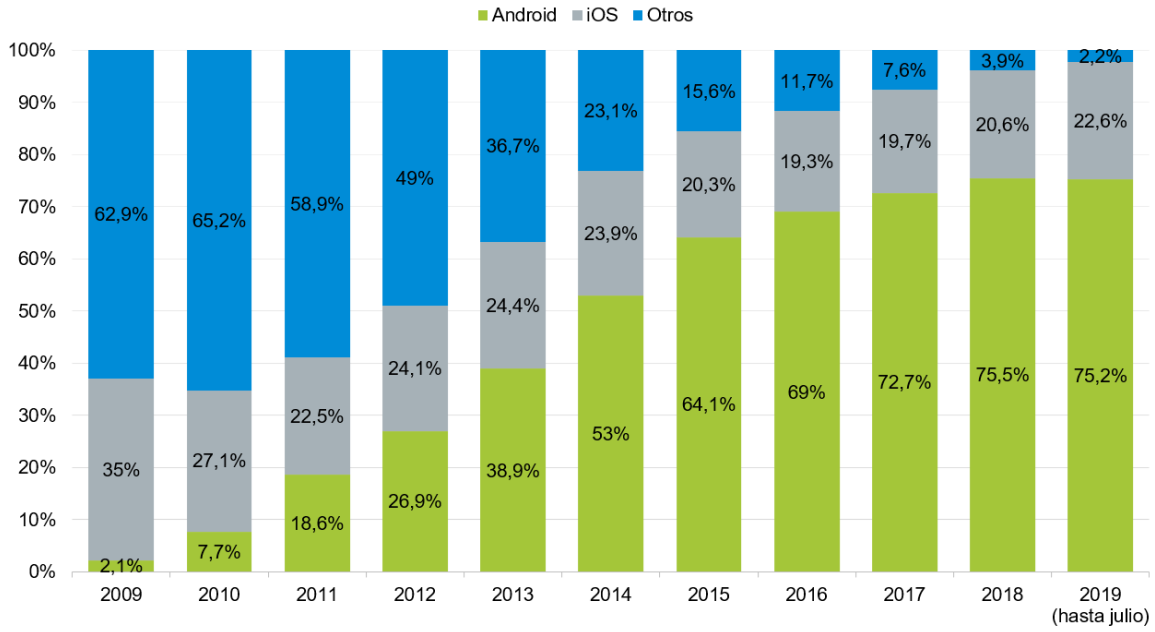

Figura 3-5: Evolución de las ventas de teléfonos móviles por sistema operativo

Decidido el sistema operativo, hay que analizar qué entorno de desarrollo se va a utilizar y aquí Android Studio se presenta como la mejor opción. Se trata del entorno oficial de Android que está en constante actualización y que, además, está dejando en un segundo plano a Eclipse en el ámbito del desarrollo de aplicaciones [8]. Por lo tanto, Android Studio es el entorno idóneo para las pretensiones del proyecto y un posible progreso de este en un futuro.

Para la implementación del algoritmo de detección de objetos y el uso de la cámara del dispositivo se ha hecho uso de las librerías proporcionadas por OpenCV, que son completamente compatibles con el entorno Android Studio. Esta biblioteca se encuentra dividida en más de una decena de módulos, aunque se han utilizado únicamente tres de ellos: *android*, *core* e *imgproc*.

Por último, partiendo de la necesidad de crear máscaras y contornos de nuevos objetos e incluirlos en la aplicación se ha empleado MATLAB, ya que mediante este entorno es posible desde la binarización de una imagen hasta la realización de todo tipo de operaciones morfológicas con la misma.

4 Desarrollo

En esta sección se explicarán los pasos que se tomaron en la elaboración de las actividades que componen la aplicación y el funcionamiento de cada una ellas.

4.1 Interfaz gráfica

4.1.1 Actividad inicial

La base de cualquier aplicación de Android son las actividades. Una actividad representa una pantalla en la que el usuario puede interactuar con el teléfono. Por lo general, hay tres archivos significativos para que una actividad sea funcional: un archivo XML, que determina la interfaz de usuario; un archivo Java, que determina la lógica de la pantalla; y el archivo de manifiesto, que contiene la configuración principal de la aplicación. Por consiguiente, el diseño de la primera actividad de la aplicación se crea mediante un archivo XML.

Todos los archivos XML se componen de los elementos gráficos que se encuentran en cada ventana. Concretamente, la pantalla de inicio de la aplicación contiene un cuatro de texto, tres imágenes y tres botones. Estos se agregan en la vista previa del diseño o directamente en el código del archivo XML creado, donde se hace referencia al id, al ancho y a la altura, a la ubicación, al texto y a la imagen de cada de uno ellos. Todos estos elementos se definen dentro de un tipo de vista denominado *RelativeLayout* que se encarga de ubicar cada uno de los componentes en posiciones relativas.

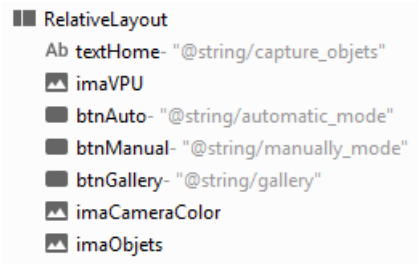

Figura 4-1: Jerarquía de vistas para el diseño de la pantalla de inicio

Una vez obtenido el diseño deseado de la pantalla inicial, es preciso especificar que esta sea la actividad con la que se inicie la aplicación. Para ello, en el archivo de manifiesto se añade en el correspondiente nodo de la actividad de la pantalla de inicio la acción *ACTION_MAIN*, que indica que es el punto de entrada, y la categoría *CATEGORY_LAUNCHER*, que indica que ese punto de entrada debe aparecer en el lanzador de aplicaciones [9].

4.1.2 Controles

La funcionalidad principal de la pantalla de inicio consiste en tener una ventana con tres botones que permitan cambiar de una actividad a otra, es decir, se pretende que mediante el pulsado de estos controles se acceda al capturado automático y al manual, y se abra la galería donde se visualizarán todas las imágenes almacenadas. Este último punto se tratará en el capítulo 4.4 de la memoria.

Con los tres botones ya agregados en el diseño XML, se procede a implementar en el archivo Java de la actividad la acción que se produce al presionarlos. Para ello, se crea tres objetos *View.OnClickListener* y se activa las escuchas llamando al método *SetOnClickListener* de cada uno de los botones. A continuación, se llama a la función *onClick*, que contiene el *Intent* que asigna, en primer lugar, en qué actividad se encuentra y, en segundo, a qué actividad se desea dirigir. Por último, se inicia la nueva actividad a través del método *startActivity*. De esta forma, cuando el usuario pulse uno de los dos primeros botones se abrirá el modo de captura seleccionado.

```
btnAuto = (Button)findViewById(R.id.btnAuto);
btnAuto.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: PantallaInicio1.this, MainActivity1Auto.class);
        startActivity(intent);
    }
});
```

Figura 4-2: Implementación de la acción del botón 'Automatic mode'

Por otra parte, para cambiar de modo o visualizar las imágenes realizadas abriendo la galería, es necesario regresar a la pantalla de inicio, lo cual se implementado mediante la creación de un nuevo botón. Este se encuentra en ambos modos de captura y cuenta con una implementación idéntica a los anteriores, pero referenciando la actividad actual y la actividad a la que se quiere desplazar, en este caso a la pantalla de inicio.

4.1.3 Icono, nombre y temas

La creación de un icono y la elección de un nombre y un estilo identificable para la aplicación es un punto importante en el desarrollo. De esta manera, facilita que el usuario se acerque a conocer el propósito de esta y la identifique entre el resto de aplicaciones instaladas en su dispositivo móvil.

Por defecto, el icono de una aplicación desarrollada en Android Studio es un androide, pero existe la posibilidad de modificarlo mediante un proceso muy sencillo. Tan solo hay que agregar el nuevo icono a las cinco carpetas *mipmap* del proyecto con una dimensión específica en cada una de ellas para adaptarse así a todo tipo de resoluciones de pantalla y asegurar que el atributo *android:icon*, situado en el archivo de manifiesto, es el título de la imagen que contenga el diseño del nuevo icono. También se encuentra en este archivo otro atributo, *android:name*, que permite cambiar el nombre de la aplicación sin necesidad de crear un nuevo proyecto.

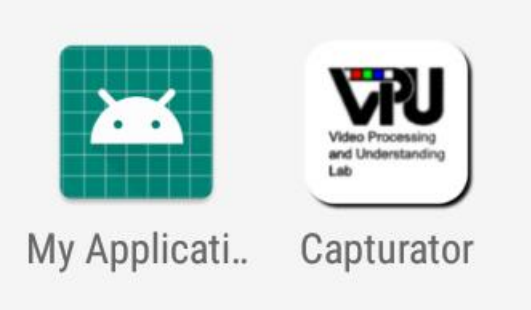

Figura 4-3: Icono predeterminado comparado con el de la aplicación actual

Para continuar con la personalización de la aplicación, Android Studio permite separar el diseño del contenido a través de un recurso XML, separado del diseño XML de cada actividad. En concreto, los dos temas que se definen en este proyecto comparten el color de la barra de aplicaciones, el de la barra de estado y el de los controles de la interfaz como campos de texto.

Sin embargo, cabe destacar que el tema que se aplica a los modos de captura cuenta con una barra de acciones en la parte superior de la ventana, que es útil para indicar si el usuario se encuentra en el modo automático o en el manual, y para acceder al menú donde se permite seleccionar las distintas siluetas de los objetos a detectar. El ejemplo visual se puede observar en la figura 3-3 de la presente memoria.

4.2 Captura de imágenes

El objetivo principal de este trabajo es la detección y captura de imágenes de objetos. La idea, por consiguiente, consiste en que el usuario sea guiado con una silueta de color rojo del objeto que se desea capturar y en el instante en el que esta se alinee con la silueta del objeto captado por la cámara, se detecte y se capture. En base a esto, se necesita en primer lugar establecer la vista de la cámara, mostrar en la pantalla la silueta fija del objeto a detectar y capturar la imagen.

La vista captada por la cámara es establecida mediante el uso de las librerías de OpenCV, concretamente con la clase *JavaCameraView*. Esta clase es una implementación de la clase *CameraBridgeView* entre OpenCV y la cámara, y solo implementa las funciones de abrir y cerrar la cámara. Asimismo, se incluyen en el manifiesto los correspondientes permisos para utilizar la cámara del teléfono y se define la orientación de la misma mediante el atributo *android:screenOrientation*.

Al mismo tiempo que se muestra la vista que capta la cámara, se establece el contorno del objeto que se quiere detectar colocando la imagen de la silueta en la carpeta *drawable* del proyecto y utilizando el método *setImageResource* en el código Java de las actividades que muestran la vista de la cámara. De este modo, se determina a qué distancia y con qué ángulo el usuario debe tomar la imagen.

```
ImageView imaContour = (ImageView)findViewById(R.id.contour);
int contour = R.drawable.contour1_red;
imaContour.setImageResource(contour);
```

Figura 4-4: Implementación de la superposición de la silueta sobre la vista de la cámara

El proceso de captura se puede efectuar mediante el pulsado del botón que carga la actividad donde se produce el capturado. Dicha actividad contiene la clase abstracta *StateCallback* en la que recibe actualizaciones sobre el estado en el que se encuentre la cámara. Estas actualizaciones incluyen notificaciones sobre la apertura, la desconexión, el cierre y sobre errores inesperados de la cámara [10]. Es en el método del primer estado donde se crea una sesión de captura que se encarga de almacenar en una matriz, con el ancho y alto adecuado, los bytes que componen la imagen que se desea obtener. Con la captura realizada, se cierra la actividad regresando a la actividad desde donde el usuario ha iniciado la captura.

4.2.1 Algoritmo de detección

Con el propósito de que se produzca el reconocimiento del objeto ha de implementarse un algoritmo de detección. El procedimiento utilizado consiste en combinar, con la operación lógica AND, los contornos de cada fotograma obtenido por la cámara con la silueta del objeto (la máscara). De esta manera, se pretende realizar posteriormente un conteo del número de píxeles blancos de la imagen resultante. Para que sea eficaz, los fotogramas y la máscara deben estar binarizados, es decir, que los valores de sus píxeles sean 0 (negro) y 1 (blanco).

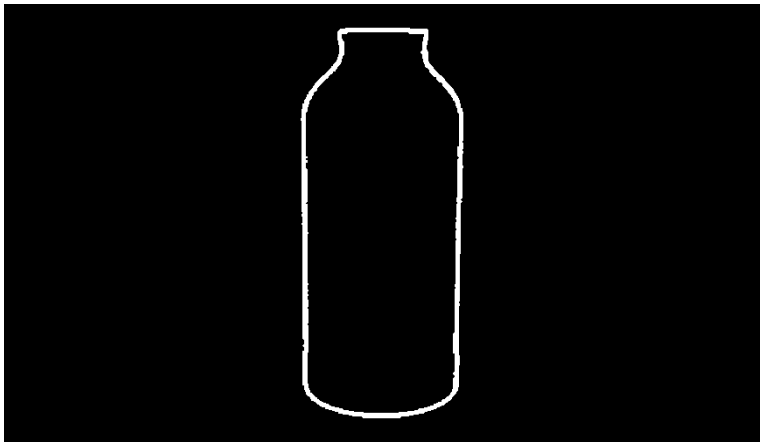

Figura 4-5: Ejemplo de máscara

La funcionalidad del algoritmo se implementa con una serie de métodos proporcionados por OpenCV en los archivos Java de las actividades de captura. Gracias al método *onCameraFrame* se obtiene cada fotograma captado por la cámara del dispositivo, lo que permite realizar todo tipo de operaciones con cada uno de ellos dentro de este método.

En primer lugar, la imagen de la cámara se almacena en formato RGB, con el método *rgba*, y se convierte, con la función *cvtColor*, a una imagen con un solo canal en escala de grises. Acto seguido, mediante las funciones *Canny* y *dilate*, se aplica el algoritmo de Canny para localizar los bordes y se realiza una dilatación con la finalidad de que los bordes detectados sean más notables.

Una vez realizadas todas estas operaciones, por medio de la función *bitwise_and*, se combina la máscara y la imagen binarizada de la cámara con los contornos localizados. Finalmente, se calcula el número de píxeles blancos de la imagen resultante a través la función *countNonZero*. En el caso de que dicho valor sea mayor al umbral escogido, se produce la detección del objeto.

4.2.2 Detección del objeto

Para detectar un objeto con la aplicación, el usuario debe superponer la silueta roja fijada en la pantalla sobre la silueta del objeto que se desea capturar. Pero, en función del modo de captura que se seleccione en la pantalla de inicio, se tomará una foto automáticamente o se modificará el color de la silueta que ejerce de guía en la captura.

4.2.2.1 Detección automática

El modo automático cuenta con una única actividad en la que, si el número de píxeles blancos calculados con el algoritmo de detección es superior al 41% del número de píxeles

del mismo color de la máscara empleada, se lanza, en primer lugar, la función que almacena la máscara en la memoria del teléfono móvil y, en segundo lugar, la actividad que se encarga de tomar la imagen y guardarla.

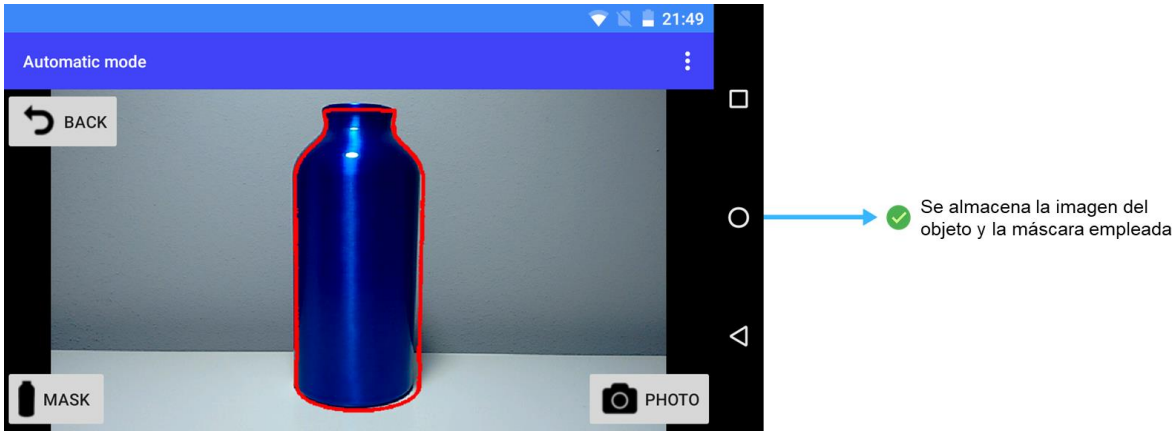

Figura 4-6: Resultado de superponer la silueta sobre el objeto en el modo automático

4.2.2.2 Detección manual

A diferencia del automático, el modo manual está formado por dos actividades. En la primera, que es con la se inicia el modo, el color de la silueta superpuesta a la imagen que capta la cámara es rojo. Sin embargo, en la segunda dicha silueta es de color verde. De este modo, cuando el conteo de píxeles blancos supere el 45% del número total de píxeles blancos de la máscara utilizada, se abre la segunda actividad en la que la silueta se cambia de color. Pero en el instante en el que el porcentaje de un fotograma se reduzca al 15%, el usuario regresa a la actividad con la silueta roja, en señal de que el objeto ha dejado de ser detectado.

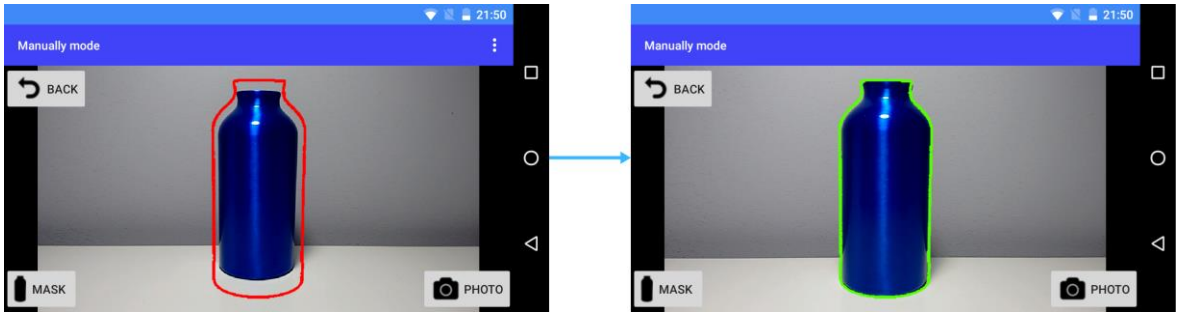

Figura 4-7: Resultado de superponer la silueta sobre el objeto en el modo manual

Cabe señalar que el umbral que ha de alcanzarse en el modo manual, para que se produzca la detección de un objeto, es ligeramente más elevado que en el automático para evitar que la silueta cambie de color de forma continuada.

4.3 Gestión de almacenamiento

Todo dispositivo Android permite un almacenamiento externo útil para guardar archivos. A fin de leer y escribir las fotografías tomadas con la cámara y las imágenes de las máscaras utilizadas en la memoria del teléfono móvil, la aplicación solicita los permisos *WRITE_EXTERNAL_STORAGE* y *READ_EXTERNAL_STORAGE* en el archivo de manifiesto.

4.3.1 Archivo de objetos

Con el propósito de guardar las imágenes de los objetos capturados al pulsar el botón 'Photo' o al superponer, en el modo automático, la silueta sobre el objeto, se crea un método en el mismo archivo Java donde se produce la captura para almacenar las imágenes en el dispositivo móvil.

Primeramente, se establece en un objeto *File* la ruta donde se almacena la imagen, que se crea a través del método *mkdirs* y que consta del directorio principal de almacenamiento interno determinado con el método *getExternalStorageDirectory*, del nombre de la carpeta de cada tipo de objeto, del nombre del archivo y de la extensión del formato de la imagen.

Cabe señalar que el título de cada imagen contiene la fecha y la hora exacta a la que se toma la foto al utilizar la clase abstracta *DateFormat*. Así se evita que al guardar un nuevo archivo se sobrescriba en uno ya almacenado. Por otro lado, las capturas de cada tipo de objeto diferente se guardan en carpetas distintas, al igual que las capturas del mismo tipo de objeto se almacenan en la misma carpeta.

Una vez establecida la ruta de almacenamiento, se escribe la matriz de bytes de la imagen en una secuencia de salida, previamente definida con la clase abstracta *OutputStream*, que contiene el objeto *File* con la ruta especificada para completar el guardado del archivo.

4.3.2 Archivo de máscaras

Además de los archivos de objetos, se almacenan las máscaras empleadas en cada captura mediante el pulsado del botón agregado 'Mask' o al detectarse un objeto en el modo automático.

Para que se produzca dicho almacenamiento, se crea un objeto de la clase *Bitmap*, el cual mediante un casting a la clase *BitmapDrawable* puede acceder a la imagen de cualquier máscara contenida en la carpeta *drawable* del proyecto. La ruta se implementa de la misma manera que en el almacenamiento de objetos, exceptuando que el nombre de la carpeta de cada máscara es distinto al de la carpeta de cada objeto, de tal forma que no se almacenen diferentes máscaras en la misma carpeta.

Posteriormente, se crea una secuencia de salida con la clase *FileOutputStream* para escribir en el objeto *File* el mapa de bits de la imagen mediante el método *compress* para que finalmente se guarde la máscara en la galería.

4.4 Galería

Para dar una experiencia más completa, se permite el visualizado desde la propia aplicación de todas las imágenes almacenadas en la memoria. Por este motivo, se agregó el tercer botón que contiene la pantalla de inicio, el cual, mediante su pulsado, permite abrir la galería del dispositivo móvil sin la necesidad de salir de la aplicación. En la galería las imágenes de los objetos y de las máscaras se clasifican en diferentes álbumes según el tipo de objeto capturado y la máscara empleada. Al mismo tiempo, el usuario puede organizar, compartir, editar y eliminar las imágenes guardadas, entre otras cosas.

La funcionalidad del botón para acceder a la galería se implementa a través del uso de la acción *ACTION_VIEW* y la selección del tipo de archivo almacenado que se quiere mostrar con *setType*. Entonces, como lo que se quiere seleccionar son fotos, se usa *image/**. Para

finalizar, se inicia la apertura, como en el resto de botones, mediante el método *startActivity*.

```
btnGallery = (Button)findViewById(R.id.btnGallery);
btnGallery.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent().setAction(Intent.ACTION_VIEW);
        intent.setType("image/*");
        startActivity(intent);
    }
});
```

Figura 4-8: Implementación de la acción del botón 'Gallery'

4.5 Inclusión de nuevos objetos

Antes de proceder a la realización de una evaluación del correcto funcionamiento de la aplicación, el proyecto exige disponer de una cantidad significativa de máscaras, utilizadas en el algoritmo de detección, y de siluetas de objetos para guiar al usuario durante la captura.

4.5.1 Creación de máscaras y contornos

El proceso de creación de máscaras y contornos se realizó mediante MATLAB. En este entorno se puede llevar a cabo la lectura, binarización, dilatación y erosión de imágenes, que son los procesos que se requieren para generar las máscaras y las siluetas.

Antes de emplear MATLAB, es imprescindible la toma de imágenes de los objetos que se quieren incluir. Convenientemente, el contraste fotográfico entre el fondo y el objeto debe ser lo suficientemente elevado para que el resultado del binarizado de la imagen sea el buscado. Por este motivo, las fotos se deben realizar en un entorno con un fondo plano de color blanco. Asimismo, es preferible que los objetos estén pintados de color negro o, en el caso de que el objeto sea un recipiente, sean rellenados con un líquido de color oscuro, como por ejemplo un refresco de cola. De esta manera, los tonos que se obtienen en la imagen son lo suficientemente cercanos al blanco y negro puro.

Una vez realizadas las fotografías, se procede con MATLAB. El primer paso consiste en leer cada una de las imágenes mediante el comando *imread*. Todas estas imágenes deben encontrarse en la carpeta de trabajo y estar en un formato soportado por MATLAB.

En segunda instancia, se convierte la imagen que se encuentra en formato RGB a escala de grises a través de la función *rgb2gray*. Con el propósito de extraer el objeto del fondo, la imagen es binarizada mediante un barrido en la matriz de la imagen que reduce cada píxel en únicamente dos valores: blanco y negro. La binarización se realiza con la instrucción *imbinarize* que, de forma predeterminada, utiliza el método de Otsu para elegir el valor del umbral. Este método escoge un umbral que minimiza la varianza intraclase de los píxeles en blanco y en negro [11].

Figura 4-9: Imagen binarizada

Para detectar el contorno del objeto que se encuentra en la imagen, se hace uso del gradiente morfológico, que consiste en la diferencia entre la dilatación y la erosión de la imagen binarizada. La operación de dilatación permite realzar la imagen, o lo que es lo mismo, que los bordes del objeto se engruesen. En cambio, la erosión degrada la imagen, causando que los bordes del objeto se adelgacen. Estas dos operaciones morfológicas se realizan mediante las funciones *imdilate* e *imerode*, respectivamente, que suministra MATLAB. Con la resta de la imagen erosionada a la dilatada, se obtiene la imagen final que se emplea como máscara en el algoritmo de detección.

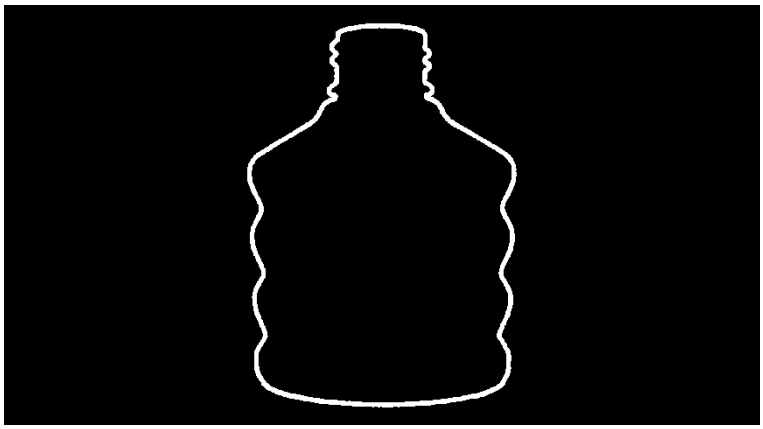

Figura 4-10: Imagen tras la aplicación del gradiente morfológico

Después de guardar las máscaras generadas, se debe escalar cada una de ellas para que tengan las mismas dimensiones que la imagen que se capte desde la cámara del dispositivo móvil. De esta forma, las operaciones que se producen entre ambas imágenes en el algoritmo de detección son compatibles.

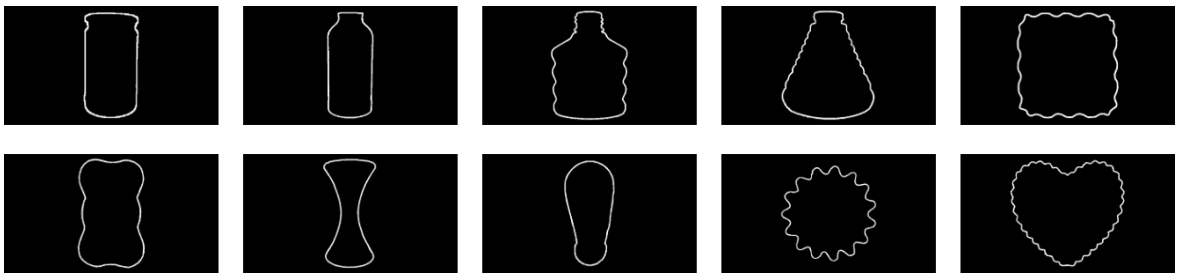

Figura 4-11: Máscaras incluidas en la aplicación

La creación de las siluetas, con el fin de que sirvan de guía al usuario, se puede realizar mediante un editor de imágenes online que permita copiar el contorno de color blanco de las máscaras previamente obtenidas y pegarlo en una imagen con un fondo transparente y con las dimensiones deseadas. Para concluir, se colorean las siluetas de color rojo y verde para incluirlas posteriormente en la aplicación.

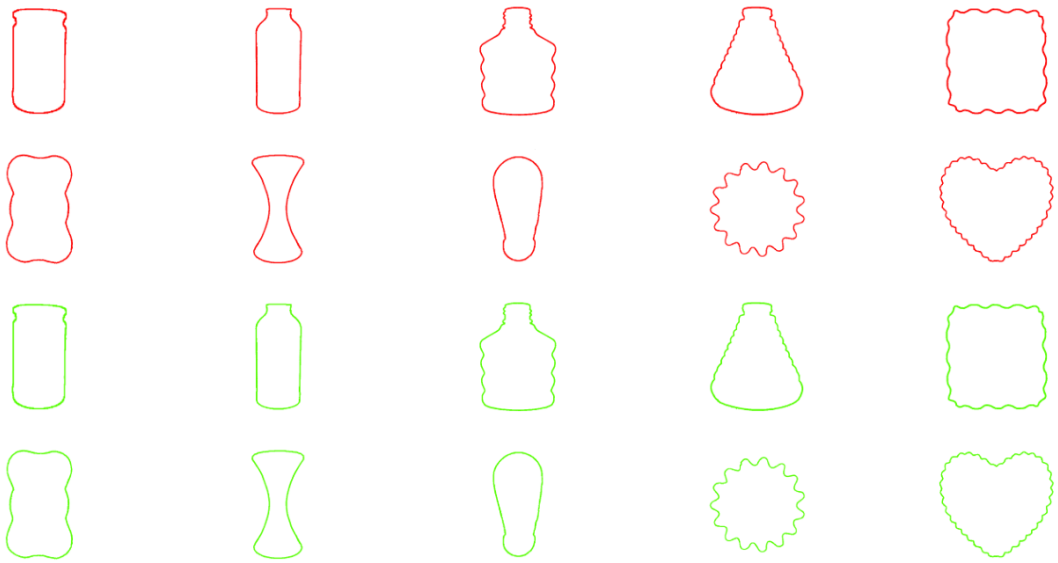

Figura 4-12: Siluetas incluidas en la aplicación

4.5.2 Inclusión en la aplicación

Todos los archivos de las máscaras y las siluetas de color rojo y verde se ubican, junto el resto de elementos de diseño, en la carpeta *drawable* del proyecto para que la aplicación pueda tener acceso a ellas. Sin embargo, es necesario crear nuevas actividades para incluir cada una de estas imágenes a la aplicación. Es importante que esta serie de actividades sean declaradas en el archivo de manifiesto como el resto, ya que de lo contrario se produce un error en el que la aplicación se detiene.

Para añadir las máscaras y siluetas que se han generado, se crean nuevas actividades, tanto de captura automática como manual. El contenido de los archivos XML de estas actividades no se modifica respecto a los anteriores debido a que la silueta y la máscara es agregada a través del código Java. Es en los archivos Java, donde se modifica el recurso que contiene las clases *R.drawable*, que son implementadas por los respectivos nombres de cada archivo.

No obstante, para que se pueda seleccionar las distintas siluetas disponibles es necesario implementar un menú para cada actividad que muestre la vista de la cámara del teléfono móvil.

Por otra parte, se deben registrar otras nuevas actividades que se encarguen del proceso de captura y guardado. En ellas solamente se adaptan los títulos de las carpetas donde se almacenan las imágenes de los objetos. Los títulos de los álbumes de las máscaras se modifican en los archivos Java descritos anteriormente. De esta manera, la galería se encuentra organizada por cada modelo de objeto y de máscara.

4.6 Menú

Para proporcionar al usuario la posibilidad de seleccionar las diferentes siluetas incluidas, se añade un menú en la barra de acciones de los modos de captura. Para ello, es imprescindible la creación de tantos archivos XML de tipo menú como actividades que muestren la vista de la cámara tenga la aplicación y la configuración del menú desde los archivos Java de estas actividades.

En cada uno de los archivos XML creados se agrega el atributo *tools:context* para que el menú se asocie con la actividad deseada. Asimismo, se indica el número de elementos que tiene cada menú. En cada uno de estos elementos se establece un identificador con *android:id* para referenciar esa opción desde el código Java y un texto con *android:title*.

Finalmente, se configura el menú en los archivos Java. Para ello, se crea, a través de los métodos *onCreateOptionsMenu* y *getMenuInflater*, cada menú en su actividad correspondiente. Además, con la finalidad de que las opciones de menú desencadenen la acción de modificar la silueta, se obtiene el id pulsado dentro del método *onOptionesItemSelected* mediante el método *getItemId*. Como resultado, se lanza la actividad asociada a ese id en función del id que pulsa el usuario mediante el método *startActivity*.

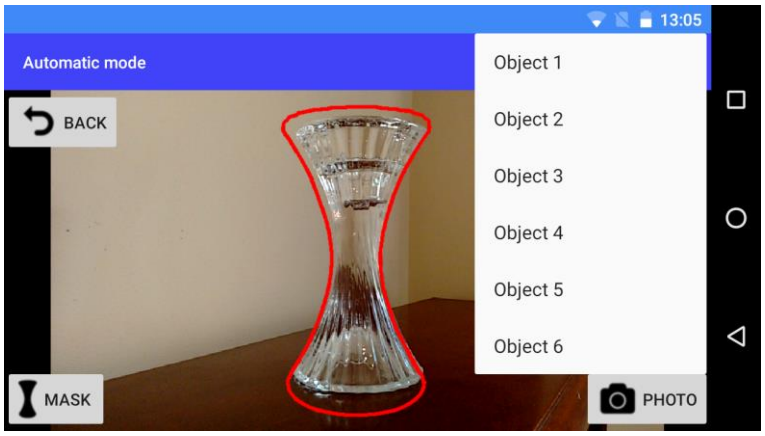

Figura 4-13: Menú de la aplicación

5 Pruebas y resultados

Para verificar que la aplicación sea plenamente funcional se procede a exponer y analizar las comprobaciones realizadas. Bajo unas circunstancias ideales se puede entrever que el algoritmo de detección implementado en los modos de captura es altamente eficaz. Sin embargo, para corroborar este aspecto, es necesario realizar una serie de pruebas que consisten en la captura de diferentes objetos en diversos entornos. De este modo, se pueden conocer con mayor exactitud las limitaciones de la aplicación y ajustar el valor del umbral del algoritmo.

5.1 Pruebas

Es necesario apuntar que tanto el objeto elegido como el entorno donde se realiza la captura condicionan notablemente la detección del primero. Es por esto por lo que es importante estudiar el comportamiento de la aplicación en las distintas situaciones que se puede encontrar el usuario.

Por un lado, la forma de la máscara escogida por el usuario puede influir positiva o negativamente en el reconocimiento de objetos. Asimismo, los objetos transparentes o translúcidos en el proceso de detección no se comportan de la misma forma que los opacos. Esto es debido a que el grado de eficiencia de binarizar un objeto y localizar sus contornos, mediante el algoritmo de Canny, depende en gran medida de si el objeto que se pretende detectar deja pasar total o parcialmente los rayos de luz que inciden sobre él o si, por el contrario, no permite pasar la luz. De igual manera, los objetos que posean una estructura lisa no tienen un comportamiento similar que en los que posean una rugosa.

Por otro lado, un objeto es reconocido de una forma más sencilla en un entorno interior que posea una iluminación controlada y un fondo homogéneo que en un entorno exterior, donde las condiciones son menos favorables para que se produzca la detección.

Por estas razones, las pruebas realizadas han consistido en numerosas sesiones de captura de cada uno de los diez objetos incluidos en la aplicación en los escenarios más comunes donde se haga uso de la aplicación. Cada sesión se ha basado en obtener el número de píxeles blancos de la imagen resultante de la operación lógica AND entre la imagen captada a tiempo real por la cámara y la máscara empleada una vez superpuesta la silueta fijada en la pantalla del dispositivo móvil sobre la del objeto que se ha deseado capturar.

Como resultado, se ha podido comprobar la eficacia del algoritmo de detección y recopilar el porcentaje de píxeles blancos en cada una de las capturas de los objetos incluidos para establecer un umbral eficiente.

5.1.1 Pruebas en entorno interior

Las primeras pruebas del presente proyecto se han llevado a cabo en un entorno interior con las condiciones más adecuadas para tomar imágenes. Los objetos que se han pretendido reconocer han sido iluminados en toda su estructura por un flexo de escritorio, que proporcionaba tanto luz fría como cálida de gran intensidad, y también por la luz solar. Además, el fondo se encontraba vacío y con el suficiente contraste fotográfico para ayudar a detectar los contornos de la imagen que captaba la cámara. Seguidamente, se muestra un ejemplo de la captura de un objeto con las características del entorno descrito.

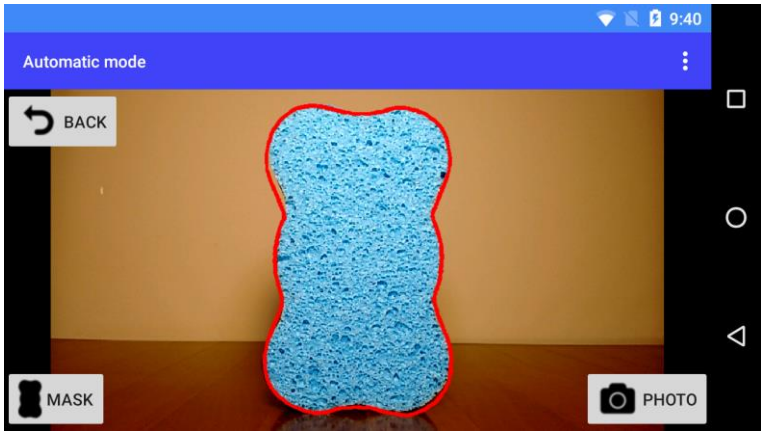

Figura 5-1: Proceso de captura en un entorno interior controlado

Posteriormente, se ha establecido un entorno con diferentes características para realizar otras sesiones de captura. Al igual que en las pruebas iniciales, se ha elegido el mismo escenario, pero no estaba tan controlado como el anterior, es decir, la iluminación no era uniforme porque dependía únicamente de la luz solar y, además, el escenario contaba con otros objetos que podían resultar ser desventajosos en la detección de un objeto. La apariencia de este entorno se puede observar en la siguiente figura.

Figura 5-2: Proceso de captura en un entorno interior no controlado

5.1.2 Pruebas en entorno exterior

Como se ha comentado anteriormente, los entornos que suelen crear más problemas en el proceso de detección son los exteriores. Por lo tanto, era fundamental que se probara la aplicación también en este tipo de entorno mediante la realización de nuevas sesiones de captura. Como se puede observar en la próxima figura, la zona exterior que se ha escogido para efectuar dichas capturas contaba con alta iluminación y numerosos elementos de fondo que podían dificultar la labor del algoritmo de detección.

Figura 5-3: Proceso de captura en un entorno exterior con alta iluminación

Por último, se han realizado unas pruebas finales en el mismo escenario anterior, con la diferencia de que la única luminosidad existente en la zona estaba proporcionada por un flexo de escritorio que emitía una luz cálida a muy baja intensidad, pero suficiente como para poder enfocar con la cámara del teléfono móvil cada objeto que se quería detectar, tal y como se muestra en la siguiente imagen.

Figura 5-4: Proceso de captura en un entorno interior con baja iluminación

5.2 Resultados

Durante las 40 sesiones de captura realizadas (cuatro para cada objeto), se ha puesto a prueba el algoritmo de detección implementado y se han recopilado los diferentes números de píxeles blancos al coincidir la silueta adherida a la vista que muestra las imágenes que capta la cámara con la del objeto a reconocer para su posterior análisis.

5.2.1 Resultados en entorno interior

A continuación, se muestra una tabla con el promedio del número de píxeles blancos de cada uno de los diez objetos que se han obtenido en un entorno interior controlado y no controlado. Como se puede observar, los cuatro objetos rugosos (objetos 4, 6, 7 y 10) son los que han alcanzado valores más altos y los cuatro objetos opacos lisos (objetos 2, 3, 5 y 9), los valores más bajos.

Objeto	Número de píxeles blancos de la máscara	Promedio del número de píxeles blancos obtenidos	
		Entorno interior controlado	Entorno interior no controlado
	7.586	5.289 (60,7%)	3.776 (49,8%)
	6.342	3.828 (60,4%)	3.021 (47,6%)
	7.552	4.283 (56,7%)	3.448 (45,7%)
	7.946	5.553 (69,9%)	4.025 (50,7%)
	10.346	6.080 (58,8%)	4.419 (42,7%)
	7.300	6.107 (83,7%)	5.289 (72,5%)
	6.699	5.105 (76,2%)	4.064 (60,7%)
	5.715	3.773 (66%)	2.946 (51,6%)
	7.367	4.059 (55,1%)	3.236 (43,9%)
	8.179	6.996 (85,5%)	6.044 (73,9%)

Tabla 5-1: Resultados en un entorno interior

El promedio del porcentaje de píxeles blancos obtenidos en los 10.000 fotogramas capturados en el entorno interior controlado es del 68,2%, un porcentaje lo suficientemente significativo como para poder comprobar que el algoritmo de detección es altamente efectivo en el proceso de captura bajo unas condiciones ideales.

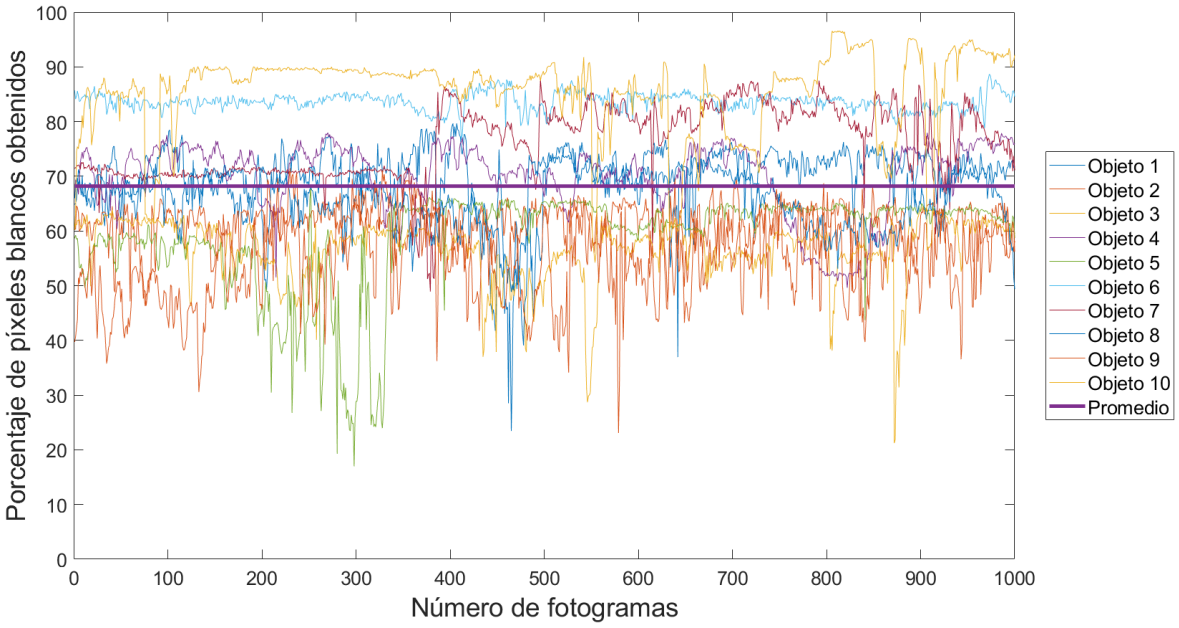

Figura 5-5: Gráfica de los resultados obtenidos en un entorno interior controlado

No obstante, el número de píxeles blancos disminuye cuando el entorno interior deja de ser controlado, pero el promedio total continúa estando por encima de la mitad, concretamente se sitúa en un 53,9%, disminuyendo menos de un 15% respecto a las primeras sesiones de captura.

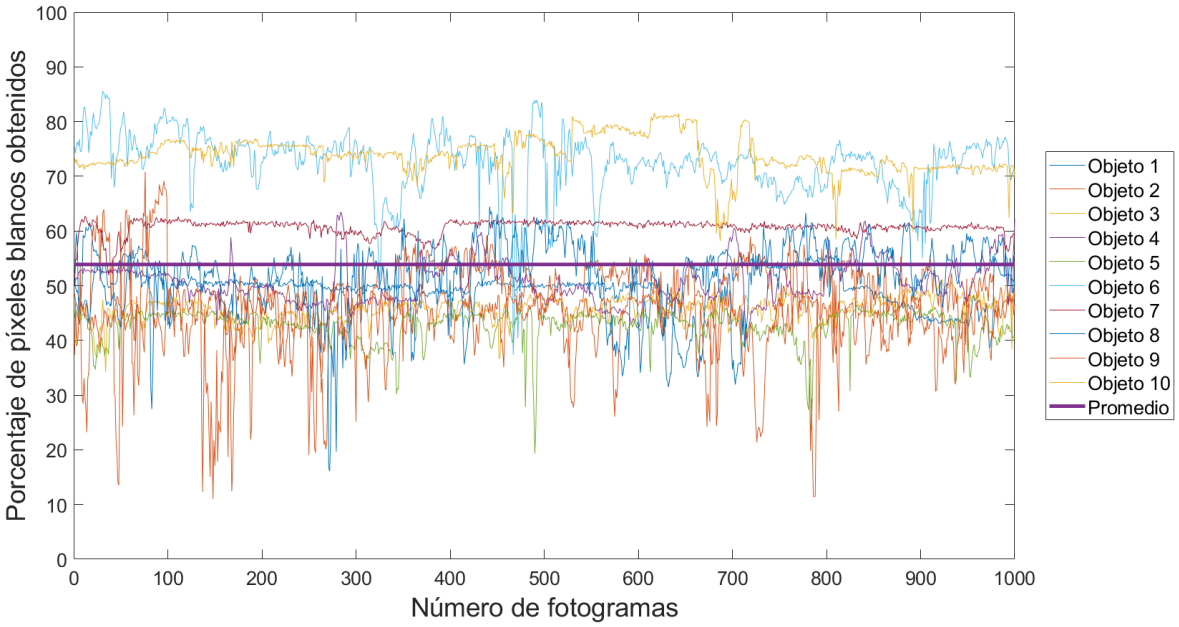

Figura 5-6: Gráfica de los resultados obtenidos en un entorno interior no controlado

Los valores más inferiores que se pueden apreciar en ambas gráficas se tratan de desajustes entre la silueta establecida en la vista y la silueta del objeto que se desea reconocer en el entorno. A pesar de que en el entorno interior no controlado cuenta con más elementos de fondo para compensar estos desajustes, la perdida de luminosidad respecto al entorno interior controlado puede producir ciertos desenfoques en la cámara del dispositivo que provocan una disminución más importante en la obtención de píxeles blancos.

Naturalmente, los mínimos más pronunciados pertenecen a los objetos opacos con una forma más irregular ya que resulta más complicado superponer la silueta de color rojo sobre la silueta de este tipo de objetos que sobre objetos con una silueta más regular.

5.2.2 Resultados en entorno exterior

La siguiente tabla recoge los resultados que se han recopilado en las sesiones de captura realizada en un entorno exterior. Al igual que en los resultados del entorno interior, los objetos rugosos (objetos 4, 6, 7 y 10) son los que mejor promedio de píxeles blancos se han obtenido, mientras que los opacos con una estructura lisa (objetos 2, 3, 5 y 9) poseen los porcentajes más bajos.

Objeto	Número de píxeles blancos de la máscara	Promedio del número de píxeles blancos obtenidos	
		Entorno exterior con alta iluminación	Entorno exterior con baja iluminación
	7.586	4.545 (59,9%)	2.785 (36,7%)
	6.342	3.632 (57,3%)	2.552 (40,2%)
	7.552	4.088 (54,1%)	3.523 (46,6%)
	7.946	5.094 (64,1%)	3.647 (45,9%)
	10.346	4.509 (43,6%)	3.468 (33,5%)
	7.300	5.529 (75,7%)	4.464 (61,2%)
	6.699	4.959 (74%)	3.579 (53,4%)
	5.715	3.606 (63,1%)	2.226 (39%)
	7.367	3.760 (51%)	2.476 (33,6%)
	8.179	6.051 (74%)	5.275 (64,5%)

Tabla 5-2: Resultados en un entorno exterior

En el entorno exterior que contaba con una alta iluminación, el promedio del número de píxeles blancos alcanzados alcanza un 61,7%, es decir, se sitúa entre el obtenido en el entorno interior controlado y el entorno interior no controlado. Con este dato, se puede llegar a la conclusión que la iluminación resulta un elemento fundamental para que el algoritmo de detección actúe con eficacia.

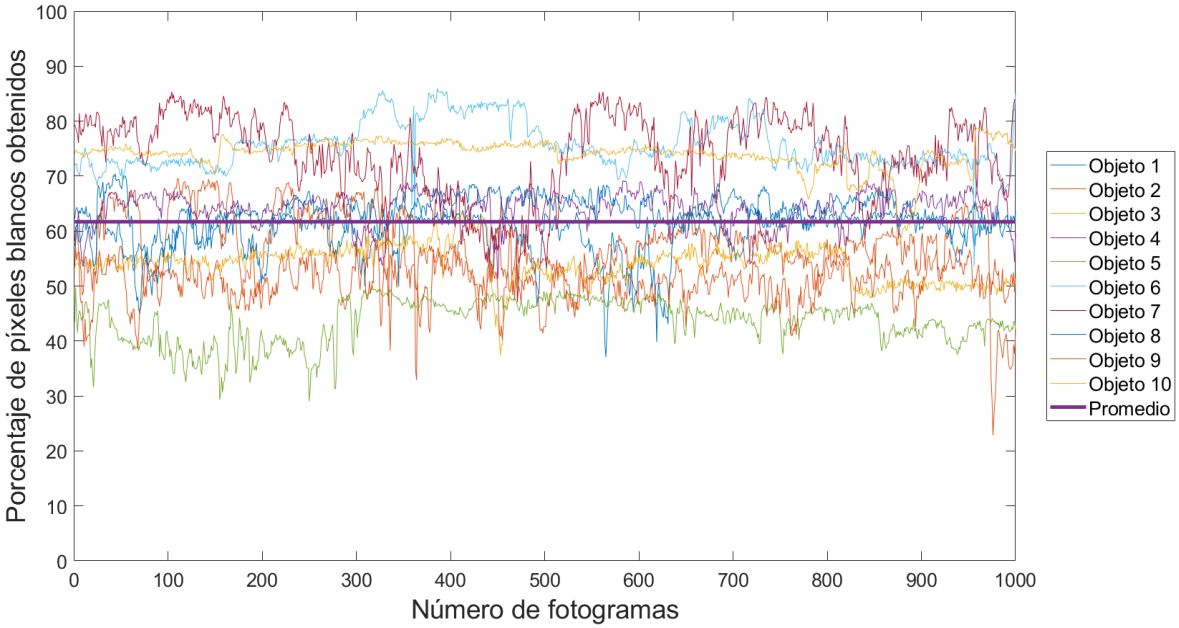

Figura 5-7: Gráfica de los resultados obtenidos en un entorno exterior con alta iluminación

En el caso del entorno exterior con baja iluminación, los resultados que se han obtenido son los más bajos de todas las pruebas realizadas. A pesar de ello, el promedio total de píxeles blancos es del 45,5%, un valor muy cercano al 50%, y solo los promedios de los dos objetos más irregulares se sitúan por debajo del 35%.

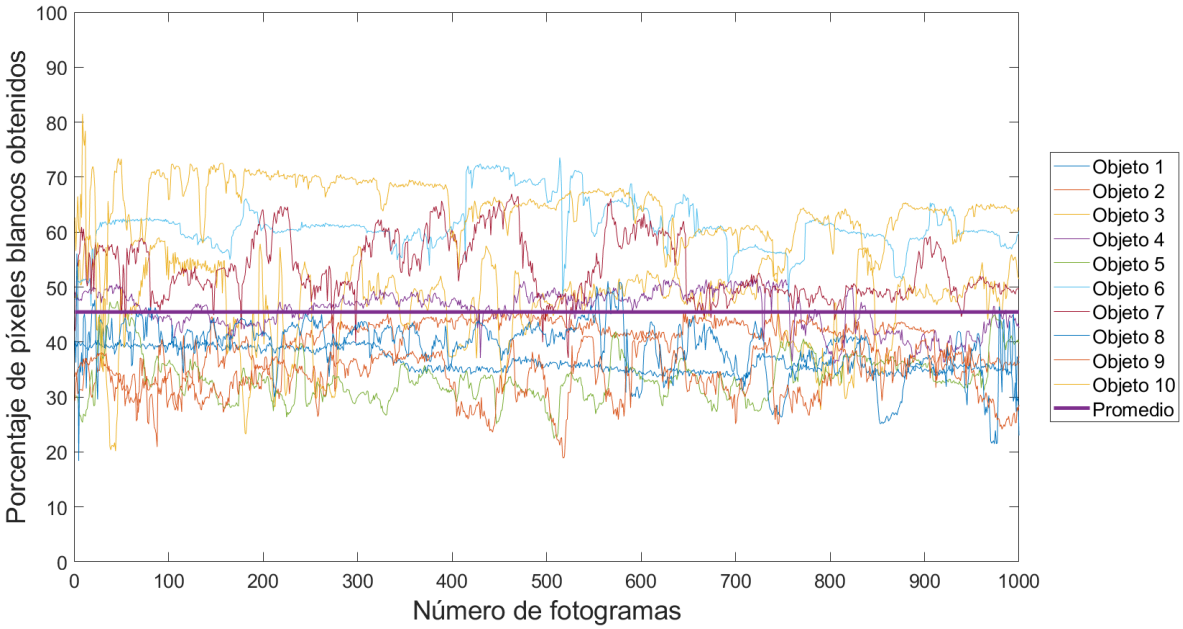

Figura 5-8: Gráfica de los resultados obtenidos en un entorno exterior con poca iluminación

Como se aprecia en las dos gráficas anteriores, la variabilidad del número de píxeles blancos es menor que en los entornos interiores, ya que los desajustes que se producen entre la silueta de color rojo y la del objeto se compensan con la presencia de otros contornos captados por la cámara. Es por eso por lo que no se encuentran mínimos acentuados con tanta facilidad como en los primeros resultados.

5.2.3 Valoraciones

Unificando los datos obtenidos de las pruebas donde se han realizado la captura de un total de 40.000 fotogramas, se pueden hallar los valores más comunes en el proceso de detección de objetos a través de la representación de un histograma que recoja el número de fotogramas donde se agrupan los distintos porcentajes de píxeles blancos alcanzados durante las 40 sesiones de captura.

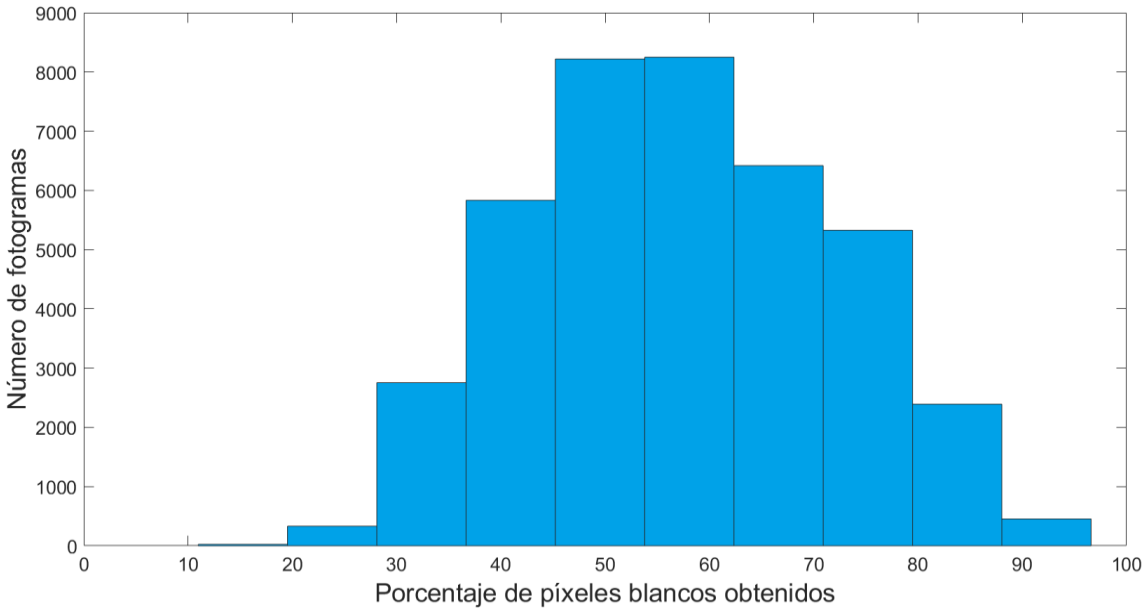

Figura 5-9: Histograma de todos los resultados obtenidos

Examinando el histograma, se observa que entre el 45,2% y el 62,3% de píxeles blancos obtenidos se concentra prácticamente la mitad de los fotogramas capturados. Sin embargo, cabe señalar que la barra que comienza a presentar una cantidad muy considerable de fotogramas es la que contiene los porcentajes que se sitúan entre el 36,7% y el 45,2%. Con estos datos y teniendo en cuenta las funcionalidades de los dos modos de captura, es razonable que el algoritmo de detección tenga un umbral diferente para cada uno de ellos.

Para que se produzca la detección de un objeto en el modo automático y, por consiguiente, se tome una foto con la cámara, el número de píxeles blancos que ha de alcanzarse debe ser superior al 41% del total del número de píxeles blancos que contenga la máscara empleada. Esto se debe a que en ese punto medio es donde comienza a contener un número relevante de fotogramas, como se muestra en la distribución de los fotogramas capturados en el histograma superior.

En cambio, si la elección del umbral en el modo manual fuera también el 41%, existiría una alta probabilidad de que durante varios instantes la silueta que guía al usuario en la captura esté continuamente cambiando de color. Por lo tanto, para que en este modo se modifique el color de la silueta con la finalidad establecida, se debe superar el 45%, ya que es a partir de este porcentaje donde se concentra el 77,7% de los fotogramas capturados en las pruebas realizadas.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

El reconocimiento de objetos es una disciplina que, pese a que ha evolucionado mucho en los últimos años, aún no se ha logrado que sea tan potente y eficaz como el reconocimiento visual de los seres humanos. Aunque existen multitud de técnicas para desarrollar aplicaciones móviles capaces de detectar ciertos objetos, la dificultad reside en elegir la que mejor se adecue a las pretensiones que desean alcanzar.

Durante el presente proyecto se ha llevado a cabo el desarrollo de una aplicación de Android con una interfaz gráfica intuitiva y funcional para el usuario, cuya intención principal es la de reconocer y clasificar ciertos objetos. En base a esto, ha sido necesario modelar previamente los objetos y detectar los bordes de las imágenes captadas mediante uno de los mejores métodos para ello: el algoritmo de Canny.

Por lo que se refiere a las numerosas sesiones de captura realizadas en entornos con diferentes condiciones, estas han resultado ser de suma importancia para establecer un valor adecuado del umbral del algoritmo de detección para los posibles escenarios que se puedan encontrar los usuarios al utilizar la aplicación.

A su vez, se ha podido analizar la eficiencia del algoritmo de detección y se ha demostrado que es de vital importancia contar con una buena iluminación para que la aplicación sea altamente fiable. Sin embargo, siempre que se cuente con una luminosidad mínima para enfocar el objeto que se desea reconocer con la cámara del teléfono móvil, el porcentaje de píxeles blancos que se obtenga será lo suficientemente alto como para que se logre detectar el objeto.

Asimismo, se han detectado otros factores que también son significativos en el proceso de reconocimiento. Por una parte, hacer uso de la aplicación en un entorno exterior puede resultar ser favorable ya que, a pesar de que los objetos se iluminan desde una posición elevada, cuentan con una alta luminosidad. Ahora bien, en estos escenarios, al haber multitud de contornos por la presencia de otros elementos, se incrementan las posibilidades de que se produzca una incorrecta detección. Por otra parte, existe una mayor dificultad en la detección de objetos que cuentan con una silueta extremadamente irregular ya que la superposición la silueta fijada en la pantalla sobre la del objeto que se pretende capturar es más complicada con este tipo de objetos.

Como conclusión, se puede afirmar que los resultados que se han obtenido cumplen las expectativas que se marcaron al inicio del trabajo. No obstante, en un futuro se podría añadir nuevas funcionalidades con la intención de perfeccionar la aplicación.

6.2 Trabajo futuro

Dado el limitado tiempo del que se ha dispuesto en la realización del proyecto, se han planteado una serie de mejoras para continuar y mejorar la aplicación desarrollada en el trabajo actual.

En primer lugar, aunque los resultados han sido satisfactorios, la eficiencia del algoritmo de detección podría crecer ampliamente si se añade a los modos de captura la posibilidad de encender el flash de la cámara y de enfocar donde desea el usuario. La implementación

de estas dos nuevas funcionalidades implicaría una subida considerable del umbral escogido y un mayor rendimiento en el proceso de detección.

Evitar falsos reconocimientos es otro posible reto de una futura versión de la aplicación. Para ello, se debería implementar la búsqueda de siluetas con forma cerrada para que, de esta manera, se eliminasen la mayor parte de contornos de posibles elementos externos.

También la funcionalidad del modo de captura manual podría ser ampliada añadiendo la opción de que la silueta que guía al usuario durante la captura se cambie a color amarillo. De tal modo que cuando el conteo de píxeles blancos que se esté obteniendo se esté acercando a superar el umbral establecido, el color de la silueta sería amarillo. Por el contrario, si el número de píxeles blancos se aleja o alcanza el valor del umbral, el color de la silueta sería rojo o verde, respectivamente.

Por último, cabe señalar que las pruebas se han realizado en un solo dispositivo móvil. De modo que, para obtener más información sobre el comportamiento del algoritmo de detección, convendría realizar nuevas sesiones de captura en teléfonos móviles con diferentes características. En este caso, habría que compatibilizar la aplicación para cada dispositivo redimensionando las imágenes de las máscaras incluidas.

Referencias

[1] “La visión por computador: una disciplina en auge”, *Informática++*, 2012. [En línea]. Disponible en: http://informatica.blogs.uoc.edu/2012/04/19/la-vision-por-computador-una-disciplina-en-auge/ [Accedido: 20-sep-2019]

[2] “Introducción a Android Studio”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/studio/intro?hl=es-419 [Accedido: 20-sep-2019]

[3] “About”, *OpenCV*. [En línea]. Disponible en: https://opencv.org/about/ [Accedido: 20-sep-2019]

[4] “OpenCV4Android SDK”, *OpenCV 2.4.13.7 documentation*, 2019. [En línea]. Disponible en: https://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/O4A_SDK.html [Accedido: 20-sep-2019]

[5] “What is MATLAB?”, *MathWorks*. [En línea]. Disponible en: https://es.mathworks.com/discovery/what-is-matlab.html [Accedido: 20-sep-2019]

[6] “Differences between Android and iOS in mobile app development”, *TheTool*, 2018. [En línea]. Disponible en: https://thetool.io/2018/mobile-app-development-ios-vs-android [Accedido: 20-sep-2019]

[7] “Mobile operating system market share worldwide”, *StatCounter Global Stats*, 2019. [En línea]. Disponible en: http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-201908 [Accedido: 20-sep-2019]

[8] “Android Studio vs. Eclipse”, *Didesweb*, 2014. [En línea]. Disponible en: https://didesweb.com/android/android-studio-vs-eclipse/ [Accedido: 20-sep-2019]

[9] “Intent”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/reference/android/content/Intent.html [Accedido: 20-sep-2019]

[10] “CameraDevice.StateCallBack”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/reference/android/hardware/camera2/CameraDevice.StateCallback [Accedido: 20-sep-2019]

[11] “imbinarize”, *MathWorks*. [En línea]. Disponible en: https://es.mathworks.com/help/images/ref/imbinarize.html [Accedido: 20-sep-2019]

Glosario

API	Application Programming Interface
APK	Android Application Package
NDK	Native Development Kit
SDK	Software Development Kit
XML	Extensible Markup Language

Anexos

A Manual de instalación

Instalación de Android Studio

La principal herramienta en el desarrollo del proyecto ha sido Android Studio. Para instalarlo, se deben seguir los siguientes puntos:

1. Se descarga el archivo ejecutable desde la página web oficial de desarrolladores de Android y se ejecuta en el equipo para comenzar la instalación.

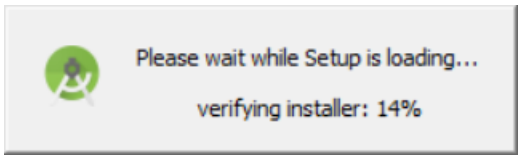

Figura A-1

2. Se inicia el instalador.

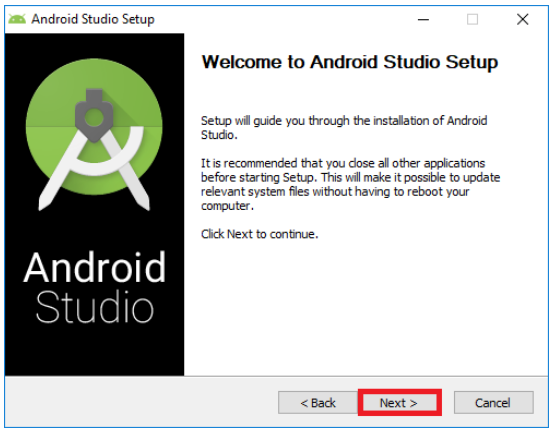

Figura A-2

3. Se seleccionan todos los componentes que se desean instalar.

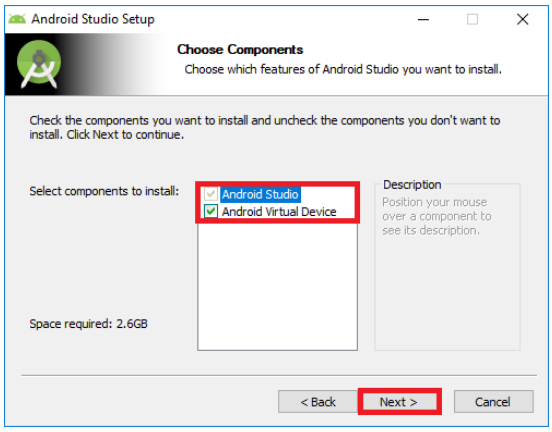

Figura A-3

4. Se indica la ruta donde se instalará el programa.

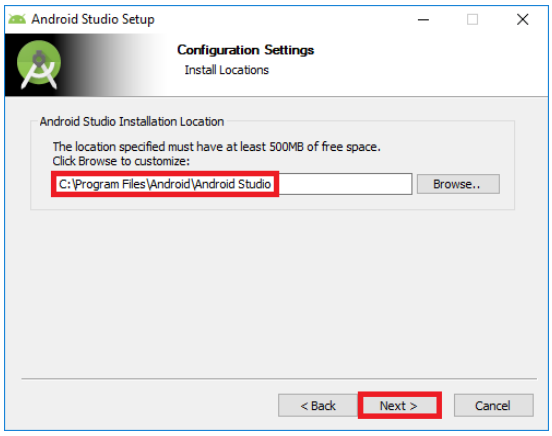

Figura A-4

5. Se indica la carpeta en la que se quieren crear los accesos directos del programa. A continuación, se inicia la instalación.

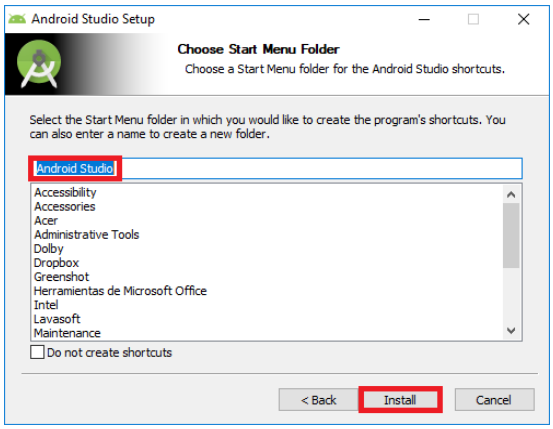

Figura A-5

6. Se completa la instalación.

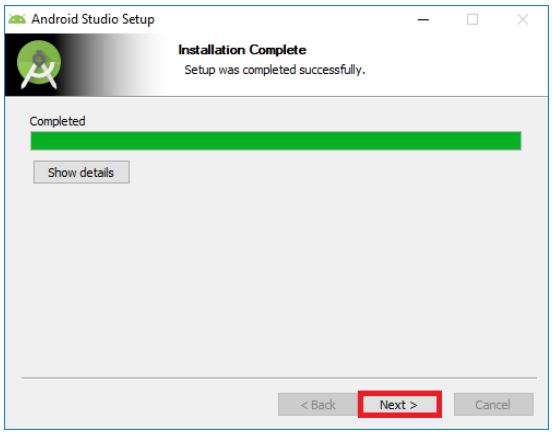

Figura A-6

7. Completada la instalación del programa, se inicia Android Studio para configurarlo.

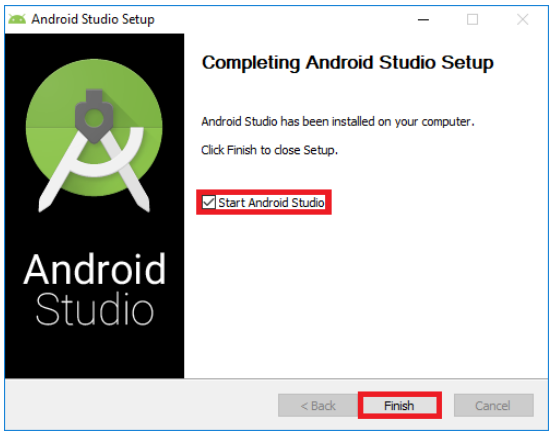

Figura A-7

8. Se selecciona la opción de no importar configuraciones previas.

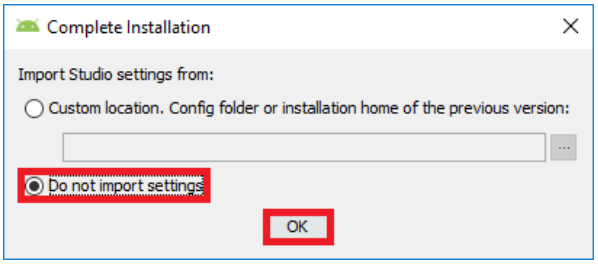

Figura A-8

9. Se abre una ventana de bienvenida que muestra todos los dispositivos en los que se puede programar con Android Studio.

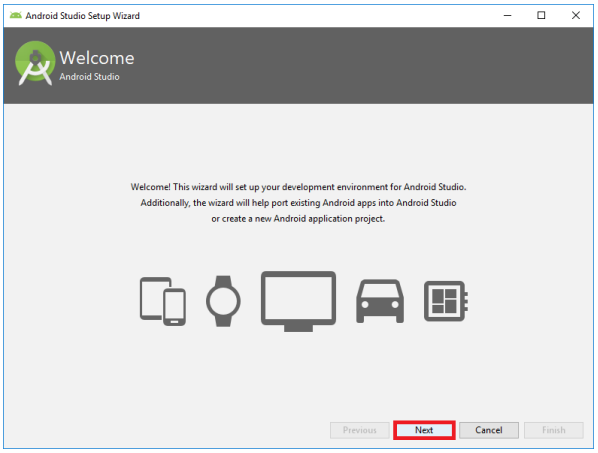

Figura A-9

10. Se selecciona la instalación estándar a fin de que Android Studio elija los componentes necesarios del SDK.

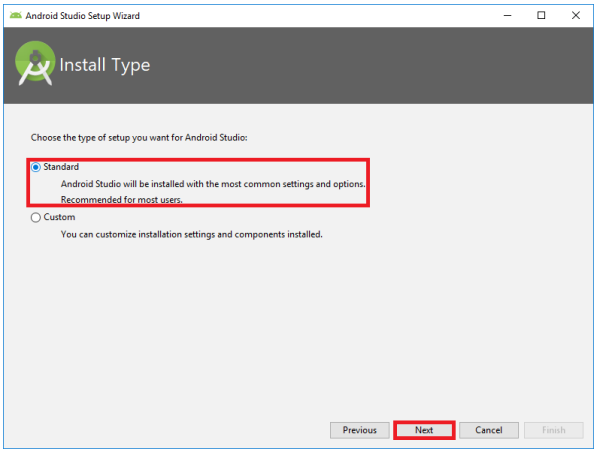

Figura A-10

11. Se selecciona el tema del editor de código.

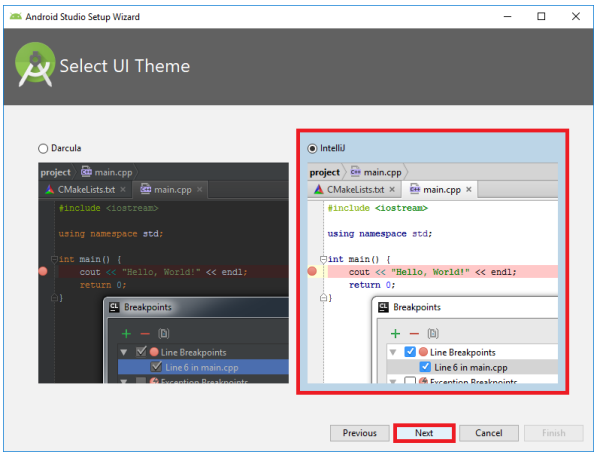

Figura A-11

12. Se procede a iniciar la descarga de los componentes del SDK.

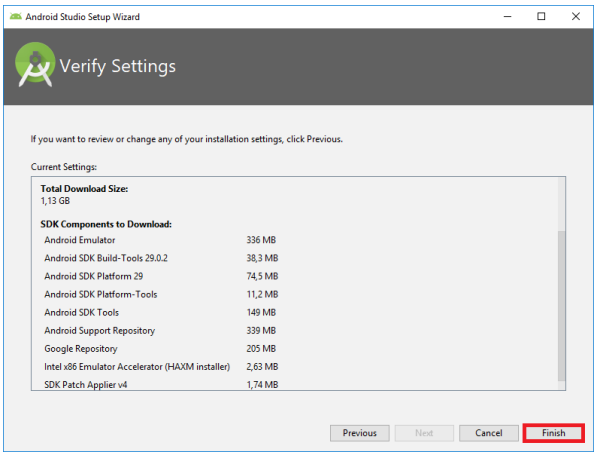

Figura A-12

13. Se completa la descarga de los componentes del SDK. De este modo, concluye la instalación de Android Studio en el sistema.

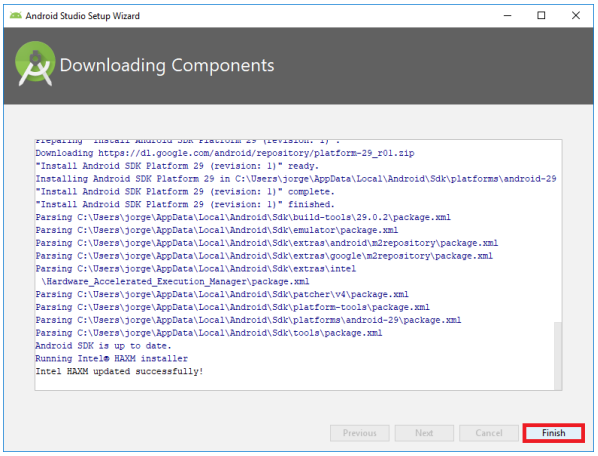

Figura A-13

Instalación de OpenCV en el dispositivo

Antes de simular el proyecto, se necesita instalar en el propio dispositivo la aplicación 'OpenCV Manager'. De lo contrario, la aplicación no encontrará las librerías de OpenCV empleadas y, por lo tanto, el usuario no podrá acceder a las funcionalidades del modo automático y del modo manual.

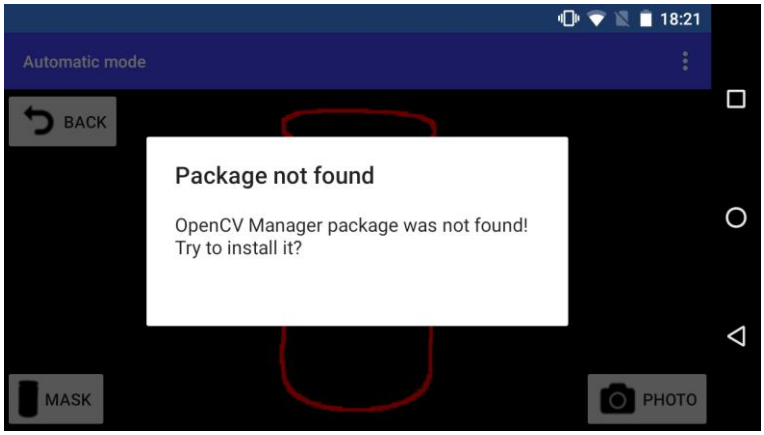

Figura A-14: Intento de acceso a uno de los modos de captura sin haber instalado previamente 'OpenCV Manager'

Para llevar a cabo la instalación de 'OpenCV Manager', es necesario completar los siguientes pasos:

1. Se descarga el archivo APK de 'OpenCV Manager' debido a que esta aplicación ya no está disponible en Google Play.

2. Se abre la carpeta de descargas del teléfono móvil y se pulsa en el APK que se quiere instalar.

Figura A-15

3. Se acepta instalar la aplicación.

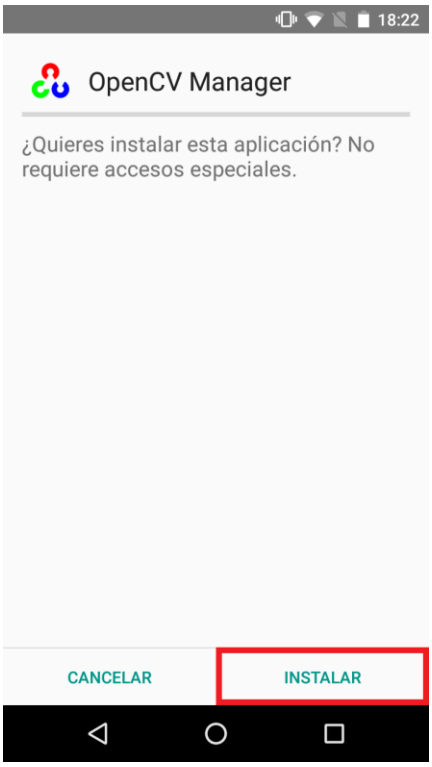

Figura A-16

4. Finalizada la instalación, la aplicación ‘OpenCV Manager’ ya está disponible en el dispositivo para ser utilizada.

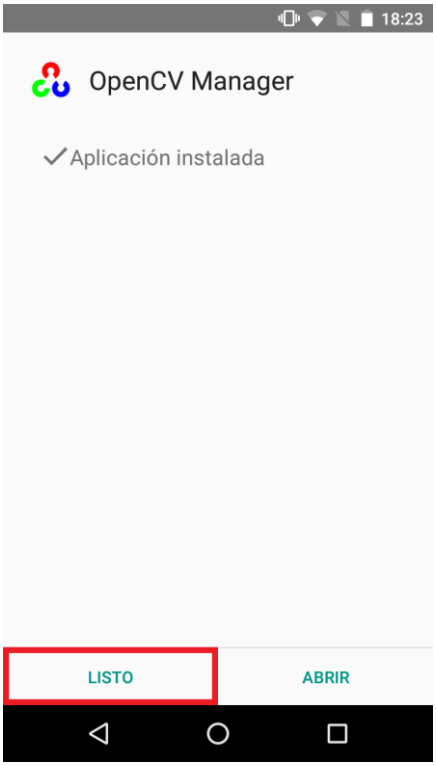

Figura A-17

B Manual de simulación

Simulación en el dispositivo

Con la finalidad de garantizar un correcto desarrollo de la aplicación, Android Studio ofrece dos alternativas: emularla en el propio ordenador de desarrollo o ejecutarla en un dispositivo físico. Dada la importancia de comprobar el funcionamiento de la presente aplicación en diferentes entornos, es conveniente que esta sea simulada desde un teléfono móvil.

A continuación, se describen los pasos que deben realizarse para ejecutar la aplicación desarrollada en un dispositivo móvil:

1. Antes de iniciar Android Studio, se debe activar la depuración por USB en el dispositivo que se quiere simular. Para ello, se requiere habilitar las opciones de desarrollo.

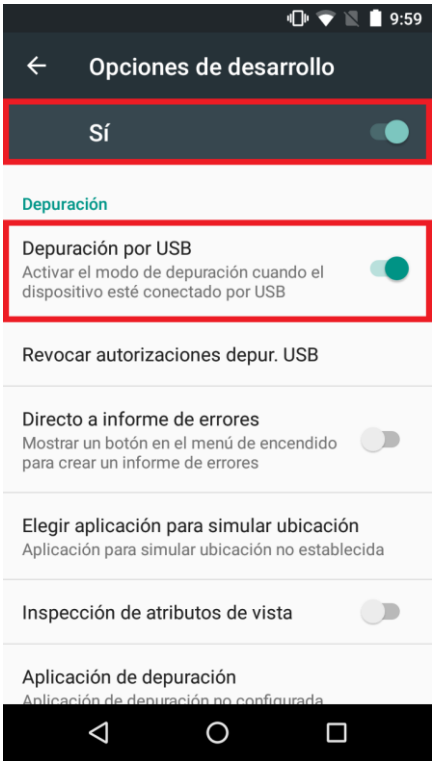

Figura B-1

2. Con la depuración por USB activada en el teléfono móvil, se inicia Android Studio y se selecciona la opción de abrir un proyecto existente.

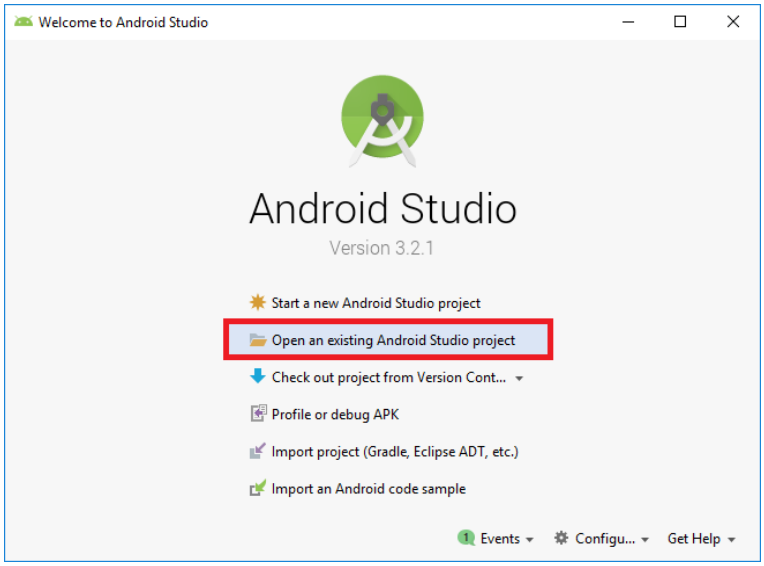

Figura B-2

3. Se selecciona la ruta donde se encuentra el proyecto para comenzar su sincronización.

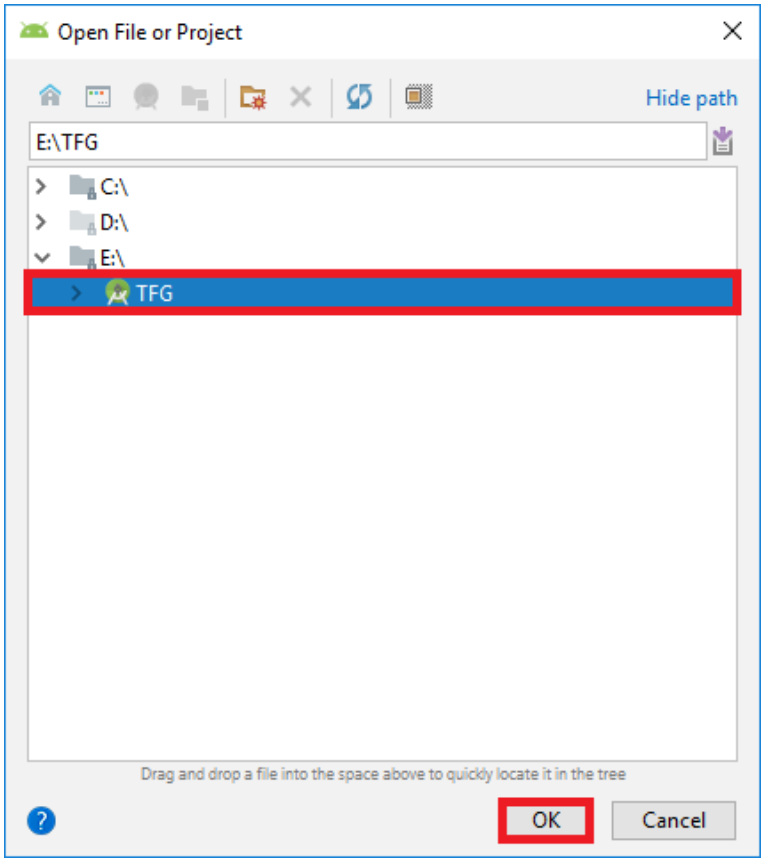

Figura B-3

4. Sincronizado el proyecto con éxito, se pulsa el botón ‘Run’ para ejecutar la aplicación.

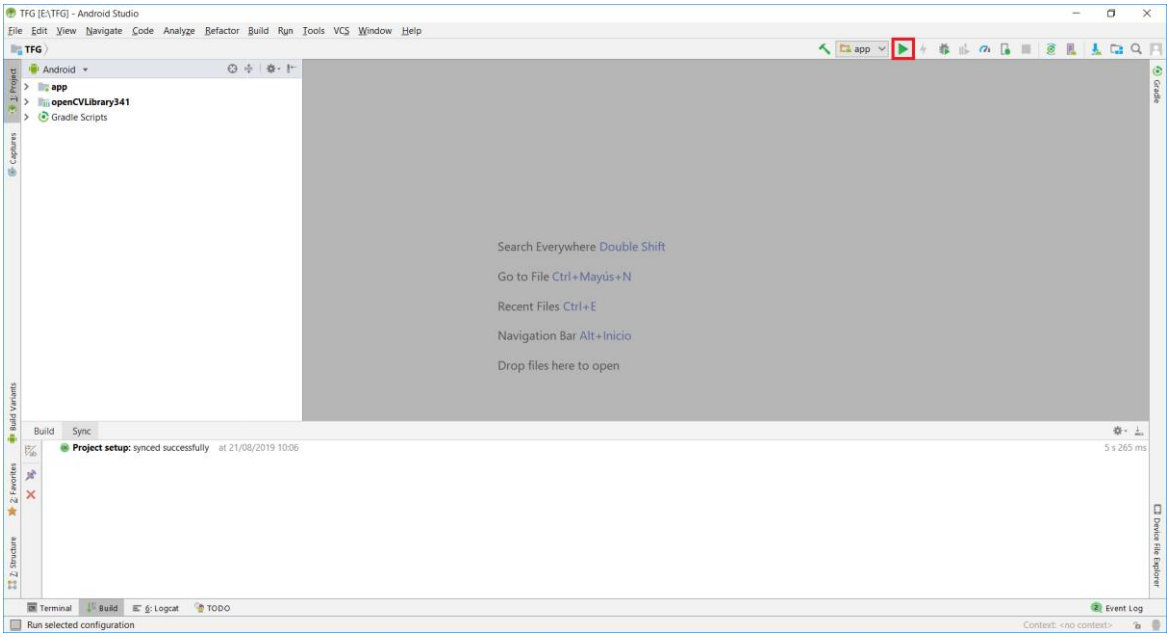

Figura B-4

5. Se selecciona el dispositivo conectado por USB. Una vez que Android Studio compile el proyecto con Gradle e instale las APKs, la aplicación se lanza en el dispositivo.

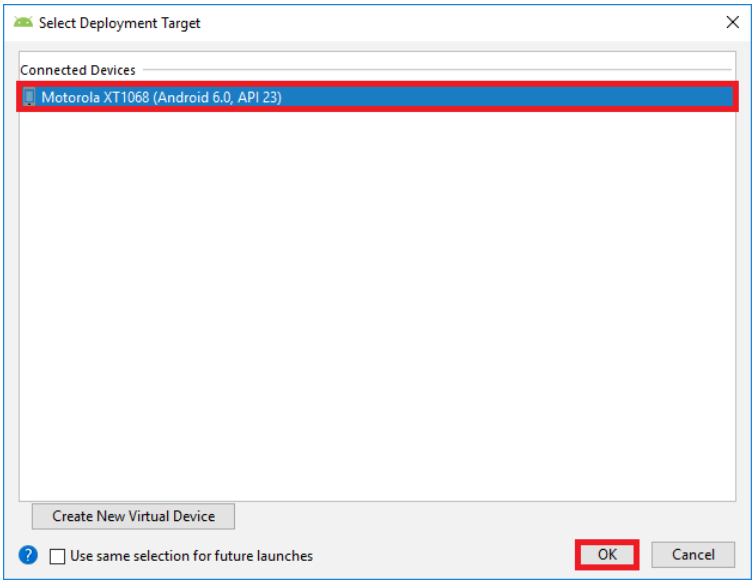

Figura B-5

Posibles errores al sincronizar el proyecto

Error de configuración

Dependiendo del equipo en el que se inicie Android Studio, existe la posibilidad de que se produzca un error de configuración durante la sincronización el proyecto de la aplicación.

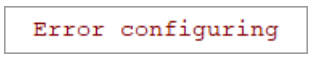

Figura B-6: Error de configuración

Para solucionar este problema tan solo se debe realizar un cambio en el archivo *build.gradle* del módulo *app* del proyecto. Concretamente, se comenta o se elimina la ruta que especifica el script de compilación *CMakeLists.txt* en dicho archivo.

Figura B-7: Resolución del error de configuración

Posibles errores al simular el proyecto

Error al instalar las APKs

Durante el desarrollo de la aplicación existe la posibilidad que se sincronice el proyecto en distintas ubicaciones del equipo o incluso en diferentes equipos, lo que puede provocar errores al instalar las APKs.

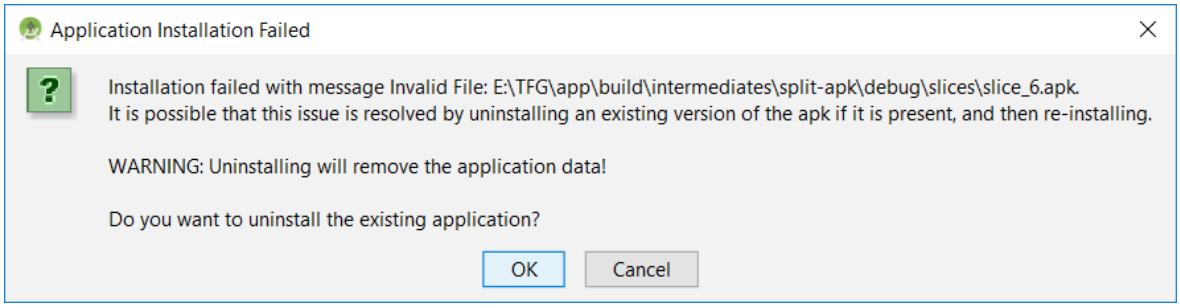

Figura B-8: Error al instalar las APKs

Lo más común para solucionar el error al instalar las APKs es asegurarse que la aplicación desarrollada esté desinstalada en el dispositivo que se quiere simular. Sin embargo, si aun teniendo la aplicación desinstalada el error permanece, se debe limpiar el proyecto, construir las APKs de nuevo y reconstruir el proyecto. De esta manera, la aplicación ya podría ser ejecutada en el teléfono móvil.

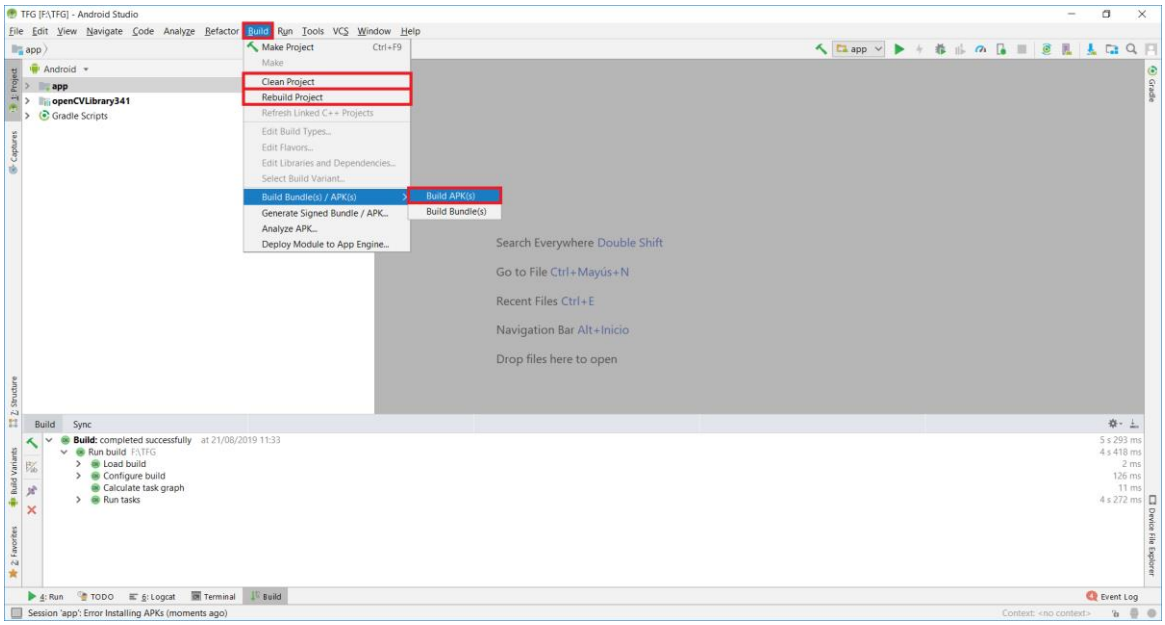

Figura B-9: Resolución del error al instalar las APKs

Error en las dimensiones de las máscaras

Si bien Android Studio proporciona todo lo que se necesita para que una aplicación sea compatible en distintos dispositivos con diferentes tamaños de pantalla, es fundamental que en el presente proyecto las imágenes de las máscaras sean las mismas que las de la imagen que se capta por la cámara ya que, de lo contrario, el uso de la operación lógica AND en el algoritmo de detección ocasionará un problema que provocará el cierre inmediato del modo en el que intente acceder el usuario. Es por eso por lo que todas las máscaras que se encuentran en la carpeta *drawable* han de ser redimensionadas en el caso que su ancho y alto no coincidan con los de la imagen procedente de la cámara.

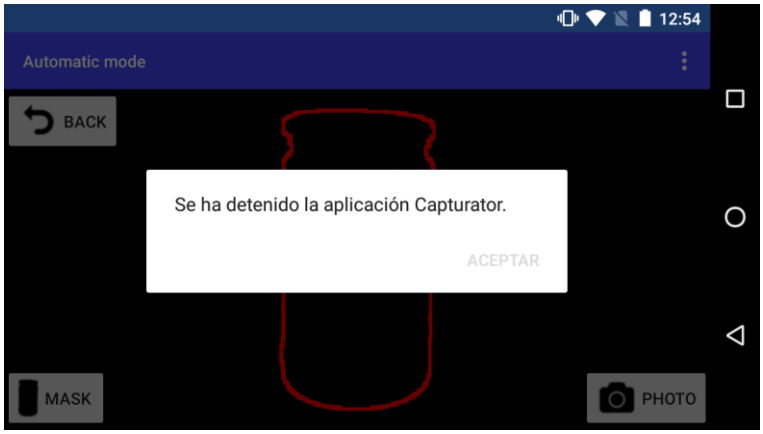

Figura B-10: Intento de acceso a uno de los modos de captura si las dimensiones de la máscara no coinciden con las de la imagen captada por la cámara

Por otra parte, es conveniente que las dimensiones de las imágenes de las siluetas de color rojo y verde también coincidan con las imágenes de la máscara y la imagen captada por la cámara con el objetivo de que el usuario sea guiado de forma correcta y el algoritmo de detección sea eficaz. Sin embargo, al contrario que con las máscaras, las siluetas no provocan ninguna incompatibilidad al ejecutar la aplicación debido a que estas se ajustan al tamaño de cualquier pantalla.